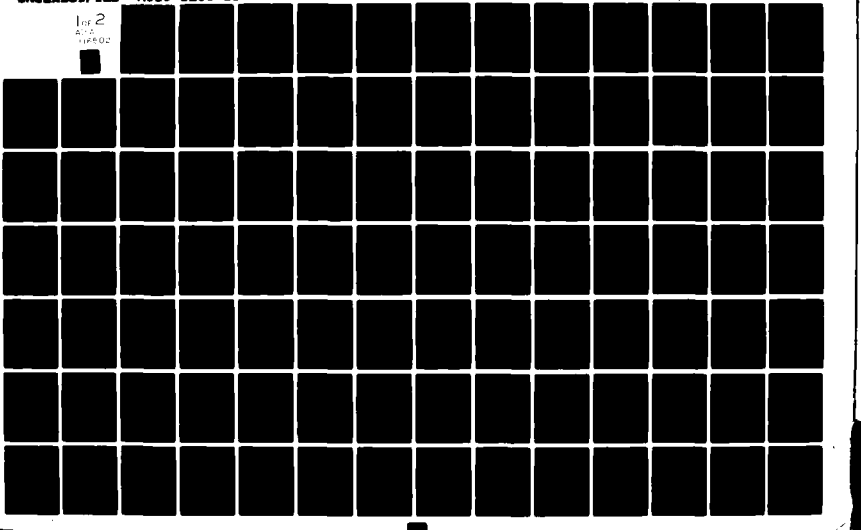
AD-A116 502 ALFRED P SLOAN SCHOOL OF MANAGEMENT CAMBRIDGE MA F/G 9/2
VIRTUAL INFORMATION FACILITY OF THE INFOPLEX SOFTWARE TEST VEHI--ETC(U)
MAY 82 J LEE N00039-81-C-0663
UNCLASSIFIED M010-8205-10 NL



Center for Information Systems Research

Massachusetts Institute of Technology
Sloan School of Management
77 Massachusetts Avenue
Cambridge, Massachusetts, 02139

Contract Number N00039-81-0663 (MIT # 91445)
Internal Report Number M010-8205-10
Deliverable Number 6

(12)

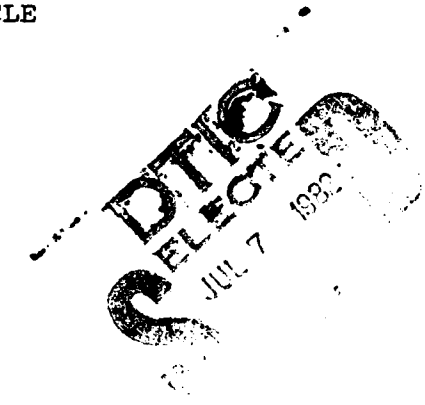
VIRTUAL INFORMATION FACILITY OF THE INFOPLEX SOFTWARE TEST VEHICLE (PART I)

Technical Report #10

By

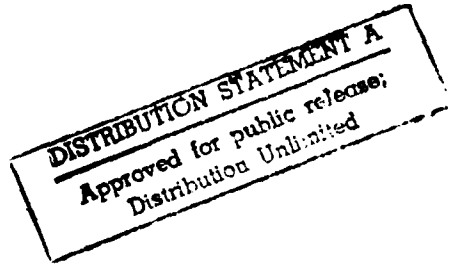
Jameson Lee

May, 1982



Principal Investigator:
Professor Stuart E. Madnick

Prepared for:
Naval Electronics Systems Command
Washington, D.C.



SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE	READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER: Technical Report #10	2. GOVT ACCESSION NO. [illegible] / 3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle): Virtual Information Facility of the INFOPLEX Software Test Vehicle	5. TYPE OF REPORT & PERIOD COVERED
	6. PERFORMING ORG. REPORT NUMBER: M010-8205-10
7. AUTHOR(s): Jameson Lee	8. CONTRACT OR GRANT NUMBER(s): N0039-81-C-0663
9. PERFORMING ORGANIZATION NAME AND ADDRESS: Sloan School of Management, MIT, 50 Memorial Drive, Cambridge, MA 02139	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS	12. REPORT DATE: May 1982
	13. NUMBER OF PAGES: 180
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office)	15. SECURITY CLASS. (of this report): unclassified
	15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

database computer, database management system, Software Test Vehicle, hierarchical system, virtual information

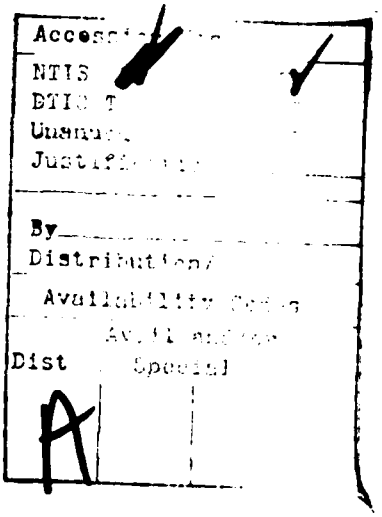
20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

This report describes the software design and implementation of the front-end for the Virtual information facility of the INFOPLEX database computer. It is part of a major effort to develop a software simulation, called Software Test Vehicle, for the underlying architecture of INFOPLEX. The virtual information facility is a single level of operations situated within the Functional Hierarchy. It supports the use of virtual information, a virtual entity based on procedural relationships and derivations from

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE
S/N 0102-014-6601

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

physically recorded data. Upon completion, this facility will be integrated within the current implementation of the STV for the INFOPLEX Functional Hierarchy which lacks the support for virtual information processing.



Virtual Information Facility
of the INFOPLEX Software Test Vehicle

by

JAMESON LEE

Submitted to the Department of Electrical Engineering and Computer Science in May, 1982, in partial fulfillment of the requirements for the degree of Bachelor of Science

Abstract

This thesis is a software design and implementation of the front-end for the Virtual Information Facility of the INFOPLEX data base computer. It is part of a major effort to develop a software simulation, so called a Software Test Vehicle, STV , for the underlying architecture of INFOPLEX.

INFOPLEX is a hierarchical architecture for data base computers, based on functional decomposition of data base operations. It is a current research project of the Information Systems Group at M.I.T.'s Sloan School of Management. Within the INFOPLEX architecture, a functional hierarchy of information management functions is built on top of a storage hierarchy of information storage functions. These two independent hierarchies are further divided into many sub-levels, each of which is devoted to a more specific function of data base activities.

The virtual information facility is a single level of operations situated within the functional hierarchy. It supports the use of virtual information, a virtual entity based on procedural relationships and derivations from physically recorded data. Upon completion, this facility will be integrated within the current implementation of the the STV for the INFOPLEX functional hierarchy which lacks the support for virtual information processing.

Thesis Supervisor: Professor Stuart E. Madnick
Sloan School of Management, M.I.T.

Contents

List of Figures

1.0.0 INTRODUCTION

INFOPLEX DATA BASE COMPUTER is a current research project of the Information Systems Group at M.I.T.'s Sloan School of Management. It proposes a new architecture whose objectives are to provide substantial improvements in information management performance over conventional computer architectures, and to provide highly reliable support for very large and complex data bases.

1.1.0 INFOPLEX OVERVIEW

Progress of modern society has put increasingly more new and challenging demands upon the capability and performance of information storage, retrieval, and management. Conventional computers, whose architecture is designed primarily for computational objectives, are not suited to meet the requirements of these new demands. Efforts have been made in four different areas to build computer systems which will suit our information needs today, and in the future: (1) new instructions through microprogramming, (2) intelligent controllers, (3) dedicated computers for data base operations, and (4) data base computers. INFOPLEX is a research project belonging to the fourth category.

1.1.1 CONCEPT

INFOPLEX employs the concept of hierarchical decomposition which organizes information management functions into a functional hierarchy, and the physical memory management functions into a storage hierarchy (Madnick 78); both hierarchies consist of many independent levels of operation, each of which supports a different set of information or storage management functions through the use of multiple microprocessors.

1.1.2 INFOPLEX ARCHITECTURE

As stated previously, INFOPLEX is an architecture for data base computers based on hierarchical decomposition. A functional hierarchy of information management functions is built on top of a hierarchy of information storage functions. Both hierarchies are further divided into many functionally independent levels of operation, each of which is to be supported by a set of micro-processors operating in parallel with one another. A global Communication Bus coordinates inter-level transmission of data. This hierarchical architecture exploits the advantages of functional modularity of operations, and of parallel processing of micro-processors to systemize data base activities and to achieve a prescribed level of efficiency. A graphical illustration of this architecture is presented in figure 1.1 .

INFOPLEX Architecture

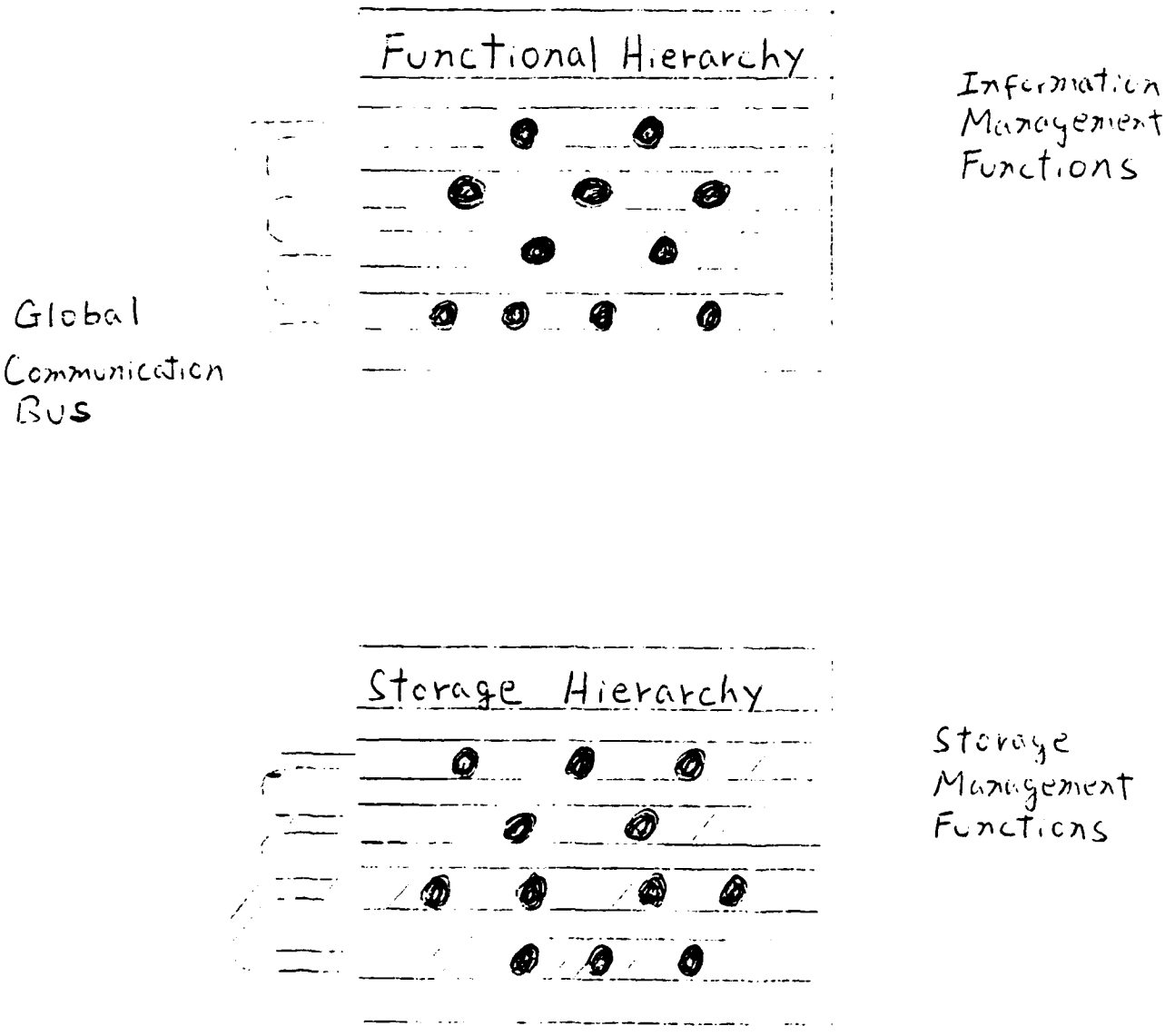


Figure 1.1

1.1.3 FUNCTIONAL HIERARCHY

Current architecture of the functional hierarchy (Hsu 1982) with respect to data abstraction consists of four separate levels: (1) external level, (2) conceptual level, (3) entity level, and (4) internal level. A part of the conceptual level is a virtual information facility (Hsu 1982). Thess four levels of information management are highly independent of one another, and each is responsible for a different but necessary phase of information processing in a data base computer.

1.1.4 *RESEARCH ISSUES*

Major efforts of INFOPLEX research are devoted to the design, modeling, and evaluation of an optimal decomposition strategy for both the functional and memory hierarchy of information management and storage operation, and also to the study of an associated distributed control mechanism. This control mechanism would be used to coordinate the activities of and inter-level communications within the hierarchies.

1.2.0 THESIS OBJECTIVE

This thesis shares a joint mission with a concurrent thesis by Peter Lu. The two theses are entirely separate in

functionalities, but closely related and dependent upon one another for a complete software simulation of the virtual information facility on the INFOPLEX data base computer architecture. This facility would incorporate the design and implementation of two sub-levels of the INFOPLEX functional hierarchy, the virtual information level, and an user interface level which is tailored for the use of virtual information processing.

This thesis is responsible for fullfillment of the front-end objectives of the joint mission; the front-end objectives include the design and implementation of the following:

a) A data base language to support virtual information

b) A finite state machine to parse data base statements written in this language

c) A user-interface tailored to the use of virtual information.

d) A processor to process the creation, listing, and modifications of virtual definitions, as well as the substitution of these definitions into data base statements in actual use.

This processor would also be responsible for transforming data base statements into a chain of tokens, each of which would include an indicator describing the classification of the token according to a prescribed classification scheme.

The combined objectives of this "front-end" and Peter Lu's "back-end" would fullfill our joint mission as mentioned earlier, namely, to construct in software a virtual information facility with its own user interface, from here on referred to as VIFI, Virtual Information Interpreter.

1.2.1 BACKGROUND

In the three short months in which VIFI was develped, we labored and wished to exhibit a certain degree of professionalism in its design and implementation. The merits of modular programming, of innovative algorithms, of performance efficiency, of functional capabilities, of user-friendliness of the proposed data based language, of program organization and flexibility, and even of consistencies in programming style were evaluated against time and labor limitations. A serious attempt was made to incorporate all of these characteristics into our Virtual Information Interpreter.

While making these considerations, many sleepless nights of unceasing arguments plagued the two developers; it was the intrinsic dissention between the idealist and the pragmatist. At a certain point, such disagreements grew to be so severe that it appeared to have left an unpleasant mark on a very close and strongly bonded friendship. However, a lesson of humanity was learned from this experience, and our cherished friendship would continue to grow, and become stronger than never before, because we have acknowledged *a feeling of faith* and destiny which was manifested through this experience. I am expressing this sentiment here because I consider it the most personally meaningful and lasting reward of this thesis.

2.0.0 VIRTUAL INFORMATION

2.1.0 Concept

The concept of virtual information in data base systems has been developed and examined in earlier research of the Information Systems Group. Basically, there is a spectrum of the kinds of information which may be retrieved from a data base. Along this spectrum, pure data occupy an extreme on one end, and pure algorithms occupy the extreme on the other. In between these two extremes are the information which may be derived from a combination of data and algorithms; such information are dynamic and procedural in nature, and are referred to as Virtual Information.

2.2.0 CLASSIFICATION

Virtual information may be categorized into three major classes: factored facts, inferred facts, and computed facts. Together, these three classes of virtual information and combinations there of, constitute the portion of the information spectrum between the two extremes of pure data and pure algorithms.

2.2.1 FACTORED FACTS

Factored facts, subsets of data elements, based on certain prescribed conditions, or so called predicates, of attribute values, are often very valuable in structuring information in a useful manner. For instance, if a certain data base maintains records of weight, hair color, and salary for a group of employees, it may be useful to select from this group those individuals who share a certain condition on their attribute values, such as having black hair, making a salary greater than 8 dollars per hour, or weighing over 300 pounds. It is important that users of information should be able to access information independent of the particular factoring involved; this would imply the ability to support multi-level factoring, or repeated factoring of data.

2.2.2 COMPUTED FACTS

Computed facts are those information which are obtainable through the application of particular computational algorithms and operators on data or groups of data. These operators include arithmatic, comparative, boolean, and other kinds of functions. In the very least, computed facts include those pure data manifested in a different form, with a different unit of measure, or an alias name. For instance: a user may define a virtual age attribute to be the difference between the current year and a person's birth-year, a virtual rectangular area

attribute to be the length multiplied by the width, or an attribute value in the unit of inches to be 12 times the attribute value in the unit of feet. In this sense, transformations between different units of measure are intrinsic to the operations of computed facts.

2.2.3 INFERRED FACTS

Inferred facts pertain to implicit relationships which the data base system may arrive at through certain levels of indirection. In other words, a path, although indirect, does exist which leads to the desired data in storage. There are two ways by which the system on its own can support this kind of virtual information. The first method is by an exhaustive search of all possible paths, and the second is the application of a certain degree of artificial intelligence to deduce a viable path to the target data. Well, the first method is unbounded in computing time, and even when a path is found, it may not be the correct path; the second method is far fetched at this time. Therefore, we will give our attention to a different but comparable set of inferred facts which is implementable, and we give it the name Pseudo Inferred Facts. Pseudo Inferred Facts are exactly the same as inferred facts except that all the indirections will be explicitly designated by the user. With this strategy, exhaustive searche is not necessary, artificial intelligence is not necessary, and the specified path would

always be the designated and correct path. For instance, the Uncle relationship may be defined as the application of the Brother relationship after the application of the Mother relationship.

2.3.0 SPECIFICATION

Users of information, through the virtual information facility, define their own working environment and the manner in which they would like to use the physical and underlying data. Such definitions of virtual information may be accomplished through a virtual information definition language. The virtual information facility would accept virtual information definitions and their modifications in the definition language, and respond to virtual information retrieval requests through a separate virtual information retrieval language.

2.4.0 MERITS

There are several major merits in the support of virtual information in a data base system. It is dynamic in nature because its definition may be created, deleted, and modified readily; its definition applies to all instances of data where it may apply, and yet there is but only one copy of this definition stored in the system. By facilitaing the ease of

modification, it enhances data base flexibility, by eliminating redundant physical records, it contributes to more consistent data, and by being procedural in nature, it enhances information accuracy through the delay in the evaluation of data which vary over time or other changing factors until their time of use. These kinds of merits are based on virtual information's association with procedural relationships. For instance: the stored algorithm for computing age would eliminate the need to update the age attribute day by day if it were physically stored, and would be applied to calculate anyone's age, thus eliminating redundancy of stored information.

Virtual information also conserves the use of vast amounts of physical storage. It makes unnecessary the storage and maintainence of those information which may be derived upon request. This raises the issue of Time/Space trade-off, which should be seriously considered when deciding which kinds of fundamental data are or are not to be physically stored. Derivation upon requests will have the added cost of derivation; therefore, those information which will be used many times and are also difficult to derive may be the best kind of data to be physically stored; those information which is seldomly used and easy to derive may be the best kind of data not to be physically stored. Furthermore, the situation is made even more complex as we realize that the definitions themselves will require the use of physical storage. Thus, it wouldn't be an

easy task to decide which kinds of data are to be derived, or to be actually stored.

The definition of virtual information on a per user basis would simulate an entire virtual data base for each individual user. Each one would be free to tailored the data base to his own preferred view or use through the virtual information definitions. A particular set of virtual definitions may be very useful for one group of users, and another set for another group of users. In this sense, each one has gotten a data base suited for his own use while not affecting anybody else's usage of the data base. A logical extension of this scenario is to implement access control mechanisms such that users may establish a controlled sharing of sets of virtual information definitions with one another; the data base administrator may monitor all such sharing to prevent unauthorized access to a certain set of virtual information functions. However, in a scenario as such, a separate catalogue would have to be maintained for each and every user, and considerable catalogue management would be required. Such is the cost for this individually user-tailored data base functionality, a secondary merit of the use of Virtual Information.

2.5.0 APPROACH

The concept of virtual information leads directly to a functional approach to data bases. A virtual information facility would be treated as a collection of functions, and retrieved data would be regarded as functional values. Virtual information requests correspond to function invocations; this functional approach to information readily supports procedural relationships on which based the concept of virtual information. As a result, a virtual information facility is likely to resemble very much a language interpreter which accepts functional definitions and respond to functional invocations with specified arguments.

3.0.0 FUNCTIONALITIES

There are numerous functionalities to a virtual information facility, each of which may be implemented to a varying degree of completeness. Although it may be desirable to implement all the functionalities there are wherever possible, it may be too impractical and less than meaningful for the initial version of the implementation. Thus, we have not implemented the One Data Base per user feature of virtual information capabilities which we have described in the previous chapter. Later portions of this chapter would describe the functionalities of virtual information which we did implement; surely, not all of these implementations would be without room for further refinement, even though they already include an extensive set of virtual information capabilities.

3.1.0 UNDERLYING DATA MODEL

The virtual information facility lies on top of the entity set level of the functional hierarchy. In this level, the data base is seen as a network of entity sets and their attributes. Each entity set may have a varying number of attributes, some of them being value attributes and others being entity attributes. (Hsu 1980) The value attributes include a set of attribute values, and the entity attributes represent

relationships leading to other entity sets. Figure 3.1 briefly illustrates this model.

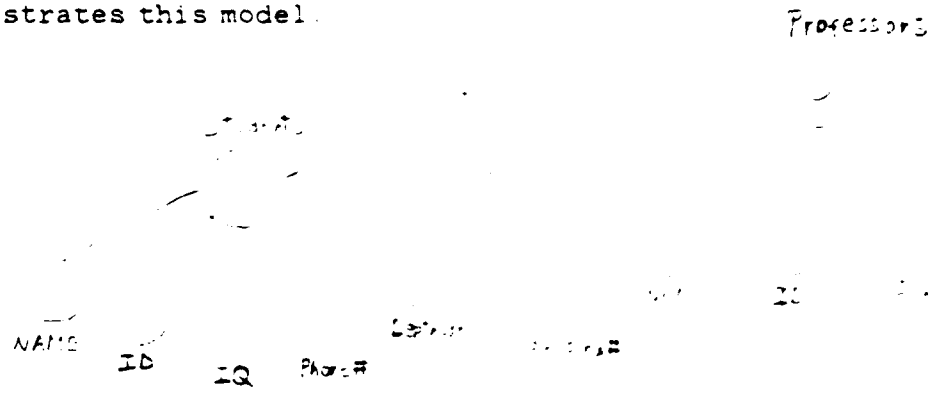


Fig 3.1

3.2.0 ACTIVE WORKSPACE

We have developed an active workspace which incorporates a line editor with full screen display, through which user commands may be issued. The workspace consists of two buffers, an execution buffer, and a transaction buffer. The transaction buffer witholds many data base statements which will be executed sequentially when the transaction buffer is executed. The execution buffer holds a single data base statement and will be automatically executed when a data base statement is completed. A number of buffer commands is created to manipulate buffer contents. The details of these commands as well as the data base statements will be illustrated in chapter 5.

3.3.0 PERMANENTLY DEFINED VIRTUAL INFORMATION

Permanent virtual information may be defined through the Define statement. Such definitions will be stored in a global dictionary, or so called catalogue, in the form of character string, and will remain there until explicitly removed or over-written by a different definition. Examples may be found within chapter 5.

3.4.0 ADHOC VIRTUAL INFORMATION

Virtual information definitions may be derived for only the duration of a single transaction. When all statements within the transaction are executed, the adhoc dictionary would be erased. Within the transaction, adhoc definition may be created, deleted, as well as modified at any time. With this feature, each transaction would be associated with a catalogue of its own, and would not interfere with the concurrent activities of other transactions executing in parallel. At this stage, we do not support concurrent transactions, but adhoc definition capability is still useful in the principle of transactions. Surely, the permanent dictionary would also be accessable from within each transaction.

3.5.0 NOTION OF A TRANSACTION

A transaction is a body of executable statements joined together within a single context. This context is provided by the adhoc dictionary associated to the particular transaction. A transaction is created within the transaction buffer, and will remain there until it is explicitly over-written, erased, or executed. Merits of this transaction concept are threefold: a) a group of statements which collectively does a certain task may be consolidated to exhibit logical unity. b) a shared context may be created and maintained for each transaction, a sign of transactional modularity and independence from one another. c) the execution of the consolidated operations in a transaction may be put off until a more opportune moment, by which time new permanent or adhoc virtual information definitions may be defined either to supplement or to replace existing definitions.

3.6.0 VIRTUAL ATTRIBUTES

Virtual attributes equated to the results of computational algorithms acting on available data or of designated indirect references may be explicitly defined through the Define data base statement. This feature incorporates the support for Computed Facts as well as for Pseudo Inferred Facts. For instance, the following is the definition and usage of two virtual attri-

butes, income and ship-country, a computed fact, and a pseudo inferred fact.

```
Define income as salary - expenses ;
Retrieve ({teachers}) by ({V0} name,income) ;
```

The foregoing retrieve statement returns two vertical columns of data. The first column being teacher's name, and the second column being their corresponding incomes.

```
Define ship-country as [redacted] company ( country (name )) ;
Retrieve ({ship}) by ({v0} name, ship-country) ;
```

This foregoing retrieve statement returns two columns of data, the first being individual ship names, and the second being the name of the country to which the ship belongs to. The entity diagram for this scenario is as follows:

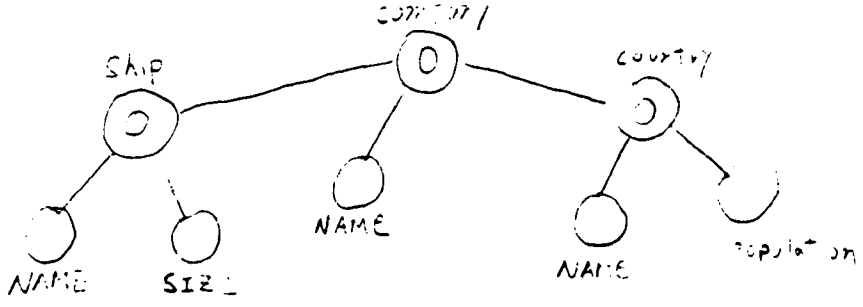


Fig 3.2

3.7.0 CONDITIONS ON REAL OR VIRTUAL ATTRIBUTES

Arbitrary conditions on real or virtually defined attributes may be defined by INFOPLEX users as the shared 'condition' on their data values from which factored facts may be later constructed. For example:

```
Define old as age > 70 ;
Define rich as assets > 1000000 ;
Retrieve ({people}where(rich and old)) by ({V0} name);
```

The foregoing retrieve statement would return a list of names of those people whose age > 70 and assets > 1000000.

3.8.0 VIRTUAL ENTITY SETS

Aside from virtual attributes, we also support a basic notion of virtual entity sets. We recognize two kinds of virtual entity sets:

a) Union or intersection of real or previously defined virtual entity sets based on their real and virtual attribute values.

b) Subsetting of real or virtual entity sets based on certain conditions on their real and virtual attribute values.

For instance:

Define ClassAB as {ClassA} MU (Name) {ClassB} ;

ClassAB is defined as the result of a multiple-union operation on entity sets ClassA and ClassB, based on a common attribute called Name.

Define RichMen as {Men} where (assets > 1000000) ;

RichMen is defined as a virtual subset of the set Men, based on the values of its asset attributes.

The complete set of union and intersection operators as well as the cartesian product operator between entity sets is illustrated within chapter 5. Also, refer to chapter 5 for details of the capability to specify various conditional predicates on attribute values.

3.9.0 GENERALIZED MACRO FACILITY

Users will be able to define arbitrary definitions and to give them specific names by which the definitions may be referred to and later substituted into data base statements. In this sense, the define statement may be used not only to

define virtual attributes, virtual entity sets, but also random definitions as well even if the definitions are seemingly incoherent without the proper context. When a retrieval statement is to be executed, all words within the statement are first checked against a list of stored definition names; any matching definition would be recalled from the dictionary and put in the place of the matching definition name in the retrieval statement. Chapter 5 includes a detailed description of such usage.

3.10.0 EXTENDABLE FUNCTIONALITIES

3.10.1 USER DEPENDENT VIRTUAL DEFINITIONS

This particular functionality is not difficult to implement, but it may be unnecessary at this stage of the project. It simply would require a separate catalogue for each user which includes an access control list, proper search rules including default situations, and adequate coordination and control mechanisms to manage the various catalogues. It would increase the cost in terms of time and space efficiency. Thus, we have not included this functionality in this version of virtual information implementation. Nevertheless, if circumstances in later time are such that the support for user dependent catalogues is so desirable as to more than compensate for its cost of implementation, this functionality may be added readily.

3.10.2 INFERRED FACTS OF UNDESIGNATED INDIRECTION

Inferred facts with undesignated indirection, rather than pseudo inferred facts with designated indirection, is likely to have tremendous costs in system performance whenever it is to be implemented. As previously stated, this would require either an exhaustive search or a certain level of artificial intelligence, both of which require large amounts of resources in computing power, storage and time. Furthermore, in order to verify that the indirection the system chooses at each step along the way is correct, the user has to monitor the computer decisions interactively; this defeats the original purpose of not having the user to designate his intended path of indirection. Thus, it seems very doubtful that this functionality will ever be implemented unless the requirements for user monitoring of the decision process is somehow eliminated.

4.0.0 PROGRAM STRUCTURE

Our implementation is done with special attention to modularity. Each primary module incorporates numerous internal sub-modules whose very existence are not known nor relevant to the implementation of other primary modules. Aside from the PL/1 modules, we have designed a data base language, and a finite state push-down automaton, each of which will be categorized as a single module as well. Figure 4.1 illustrates the control structure and data flow of all the modules in our implementation. Single arrow heads in the diagram represent control structure transitions and double arrow heads represent data flow.

4.1.0 MODULE DESCRIPTION

The front-end as designed and implemented in this thesis includes the following modules:

(1) USER-INTERFACE

(2) BUFFER

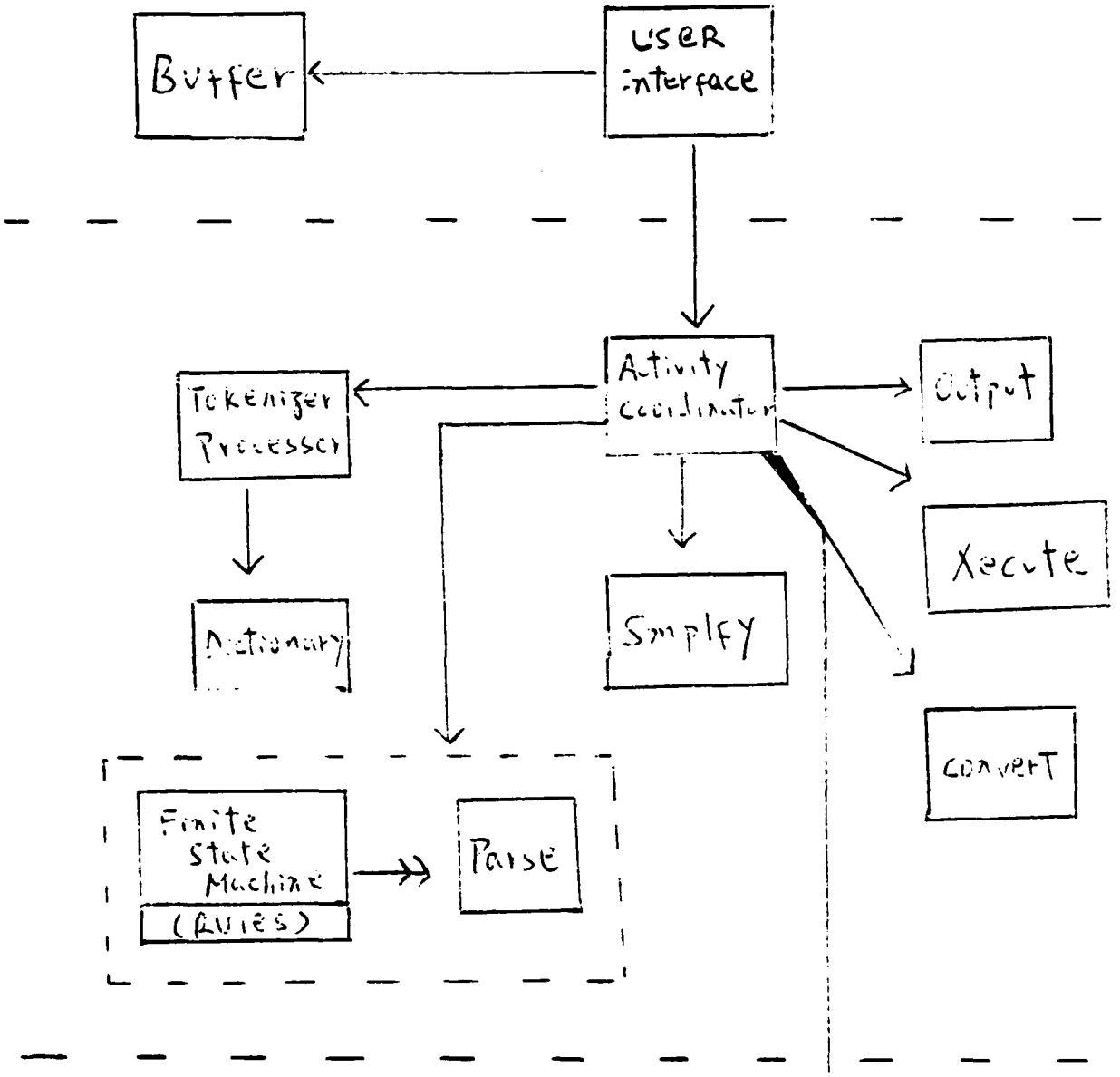
(3) ACTIVITY-COORDINATOR

(4) TOKENIZER-PROCESSOR

(5) LANGUAGE DESIGN and SPECIFICATION

(6) FINITE-STATE-PUSH-DOWN AUTOMATON (MACHINE)

Module Diagram



All programs are written in PL/1 under CMS operating system on IBM VM/370.

4.1.1 USER-INTERFACE

This module is named USINT as a PL/1 program. It is currently the options-main program of the entire virtual information facility. It diverts control to one of two INFOPLEX implementations, one of which includes our virtual information facility, and the other does not. Once the implementation with virtual information facility is selected by the user, this module serves as the communication link between the BUFFER module which interacts with the user and the ACTIVITY COORDINATOR module on the lower level which supervises the execution of data base statements.

This module has five internal routines:

(1) XBUFF

(2) GETS

(3) REPLACE (internal to RETDSPLY)

(4) RETDSPLY

The XBUFF routine strips individual executable data base statements one by one off the buffer, and pass them down to the next level for further processing.

The GETS routine is a generalized tool which actually does a substring command from the first character of a given string to the first occurrence of a given character. After the execution of this routine, the portion of the original string up to and including the given character would be eliminated.

For instance:

```
str1 = 'abc$def'
Gets (str1,'$') would return 'abc'and the value of str1
becomes 'def' .
```

The REPLACE routine is also a generalized tool to replace all occurrences of a given varying character string of length two or less, by another varying character string of length two or less.

For instance:

```
str1 = 'abc,def'
Replace (str1,',','55') would change the value of str1 to
'abc55def'
```

The RETDSPLY routine simply displays the current retrieval statement which is being processed to indicate the correspondence of subsequent outputs to this particular statement. It

makes numerous calls to REPLACE because many characters have been previously translated to enable the application of the finite state machine.

4.1.2 BUFFER

The BUFFER module is named NEWBUF as a PL/1 program. It continuously interacts with the user during a virtual information session. It has a transaction buffer which corresponds to the "transaction" concept of virtual information, and which would accumulate successive data base statements until the entire transaction is to be executed. It also has an execution buffer which would be automatically executed upon the completion of a single executable data base statement. The word "execution" in the context of this module simply means the return of control to the module which called it, USER-INTERFACE. When returning control to the caller, if the transaction buffer is to be executed, then the transaction buffer content will be moved to the execution buffer, and if user requested the termination of the virtual information session, then a control bit passed to it from USER-INTERFACE would be set.

In order to facilitate the using of virtual information, this module incorporates a full screen but simple line editor which is coupled with the existing buffer commands. Buffer commands enable the moving of data from buffer to buffer, execution of

either buffer content, input of buffer content from a CMS file, saving of buffer content to a CMS file, and termination of the active session. The editor commands are INSERT, DELETE, and TOPLINE; they enable a simple editing of the transaction buffer content. The usage of these editor commands and buffer commands is described in Chapter 5.

In essence, this module establishes the Active Workspace environment described earlier. It is the primary module of the external level of the functional hierarchy, developed specifically for the use of virtual information facility on the next lower level.

This module has the following internal routines:

(1) LDSPCH
(2) BUILDBUFF
(3) TRNSLATE
(4) FINPUT
(5) KBLKS
(6) GETS
(7) NEXTWORD
(8) RMVFBLKS (internal to NEXTWORD)
(9) FSAVE
(10) WAIT
(11) REPLACE
(12) HELP

(13) SETMKS

(14) BDISPLAY

(15) DELTE

(16) WTHNLM

The LDSPCH routine contributes to the format integrity of each user inputted line by constructing a header which is concatenated to the front of each line. The header begins with a "@" character, which is suceeded by a numeric character string representing the number of leading blank characters in this line, and ends with a ":" character. In this manner, all leading blank characters of each line may be removed. The advantages of using such a header are twofold; not only can storage be conserved, but also a fixed structure be imposed on all user inputs to reduce complexity.

The BUILDBUF routine constructs either the execution or the transaction buffer one line at a time from each user input line. Markers on either buffer is repositioned to enable proper editor display. It returns a boolean value of "true" if there is at least one completed data base statement in the current buffer.

The TRNSLATE routine checks for missing quote terminators for character string constants and back-slash terminators for comment lines. It also translates ";" characters within com-

ments to "%2" and consecutive single quote characters within character string constants, representing an actual quote character, to "%1". Such translations are necessary to avoid ambiguity and complications in the input recognition stages of the process.

The FINPUT routine serves to input transaction buffer content from a CMS file whose file name is "file" and file type is given by the user through the "finput" buffer command. Original transaction buffer content is erased. Characters "%",",",":","@" are replaced by "%4","%0","%3","%5" so that they would not interfere with finite state machine command language.

The KBLKS routine serves to remove all leading blank characters from the current input line, and also to keep the number of them removed in variable "ldspaces".

The GETS routine is a general tool as described earlier within the USER-INTERFACE module.

The NEXTWORD routine returns the sequence of characters in the input line up to but not including the first blank character. If a blank character is not found, the entire input line is returned. Comments are automatically removed and are not recognized as part of an input line.

The RMVFBLKS routine is internal to NEXTWORD; it serves to remove the blank characters preceding each word in the input line. Its name stands for remove-front-blanks.

The FSAVE routine is the counter part to the FINPUT routine. It writes the current transaction buffer content into a CMS file whose file name is "file" and file type is given by the user through the "fsave" buffer command. The characters translated by FINPUT and TRNSLATE routines are restored before they are written into the CMS file.

The WAIT routine serves as a time delay to hold messages to user on display terminals long enough to be readable by the human eye.

The REPLACE routine is a general tool as described earlier within the USER-INTERFACE module.

The HELP routine displays a brief explanation of each buffer command to the display terminal.

The SETMKS routine sets or resets markers in either the execution or the transaction buffer for buffer-display purposes of the full-screen line-editor. The name "setmks" stands for "set-markers".

The BDISPLAY routine serves to display the contents of both the execution and the transaction buffer. It implements the full-screen characteristic of our line-editor.

The DELTE routine implements the delete-line function of the editor. The transaction buffer markers are properly reset after each invocation of this routine.

The WTHNLM routine serves to verify the logical correctness of editor command correctness. If the parameters are out of current buffer boundaries, then the routine will return '0'B .

4.1.3 ACTIVITY COORDINATOR

This module coordinates all activities on the level of virtual information processing. It directs the moving of program control through various modules on this level. A number of debugging tools which prints out various trees, token chains, and tables are included within this module, and can be used in times of need by inserting a "call" statement any where within the module.

This module contains the following internal routines:

(1) GETS

(2) PRINTT

(3) PRINTM

(4) PRINTX

(5) PRINTE

(6) PRINTR

The GETS routine is a general tool as described earlier within the USER-INTERFACE module.

The PRINTT routine is a debugging tool which can be used to print the chain of input tokens.

The PRINTM routine is a debugging tool which can be used to print a snap shot of the finite state machine.

The PRINTX routine is a debugging tool which can be used to print the execution tree.

The PRINTE routine is a debugging tool which can be used to print the entity set table.

The PRINTR routine is a debugging tool which can be used to print the revised entity set table.

4.1.4 TOKENIZER-PROCESSOR

This module serves to tokenize retrieval statements, and to execute "define" , "adhoc", and "listdef" statements. It is the only module of the virtual information facility which communicates with the dictionary of virtual information definitions, besides USER-INTERFACE which makes one call to dictionary for initialization. When tokenizing each retrieval statement, virtual definitions are recalled from the dictionary whenever appropriate and substituted directly into the retrieval statement.

This module contains the following internal routines:

(1) GETS
(2) NXTKSTR
(3) RMVFBLKS (internal to NXTKSTR)
(4) TOK1 (internal to NXTKSTR)
(5) DEF
(6) BDTKCHN
(7) MSG
(8) LISTDEF
(9) DEFDSPLY (internal to LISTDEF)
(10) REPLACE

The GETS routine is a general tool as described earlier in the USER-INTERFACE module.

The NXTKSTR routine is the core of the tokenizing process; it recognizes from the input stream the next token in the form of a character string. Each token is a separately recognizable entity. This routine is called repeatedly by the BDTKCHN routine which builds an entire chain of tokens.

The RMVFBLKS is the same routine as described in the BUFFER module.

The TOK1 routine is the main body of the NXTKSTR routine. It recognizes the next portion of the input string, which is to be transformed into a separate token.

The DEF routine serves to execute the "define" and "adhoc" data base statements. It creates and modifies virtual information definitions in the dictionary of virtual definitions.

The BDTKCHN routine builds an entire chain of linked tokens. Each retrieval statement is transformed to such a chain of separate tokens before further processing.

The MSG routine outputs a message line to the terminal and prompts the user to press the "enter" key to continue.

The LISTDEF routine executes "listdef" data base statements. It would recall the definition in the dictionary which is to be listed, and output the definition to the terminal.

The DEFDSPLY routine is internal to the LISTDEF routine. It serves to process a stored definition for terminal display. Retranslation is needed to reconstruct those original characters which have been previously translated.

The REPLACE routine is a general tool as described in the BUFFER module.

4.1.5 LANGUAGE DESIGN and SPECIFICATION

This module incorporates the design and formal specification of a data base language which defines and retrieves virtual information. Chapter 5 is devoted exclusively to explaining and describing this module.

4.1.6 FINITE STATE AUTOMATON (MACHINE)

This module serves to parse the data base statements written in the language illustrated in Chapter 5. It consists of a set of Match-Action-Nextstate rules which is one of the inputs to a generalized parse program written by Peter Lu. A change in the grammar of our data base language readily corresponds to

changes in the rules of this finite state machine; thus, freeing us from changing the parse program itself. Chapter6 is devoted exclusively to explain the workings of these rules.

4.2.0 INTERNAL GLOBAL VARIABLES

USER-INTERFACE MODULE:

```
execbuff   -- execution buffer
trnsbuff   -- transaction buffer
firstlast  -- passed to BUFFER and used to indicate when
              to terminate end of session.
line       -- used to hold user-input line
```

BUFFER MODULE

```
execbuff, trnsbuff, firstlast (same as in USER-INTERFACE)
prstline   -- current input line, char(80)
strnsp     -- prstline, stripped of leading and ending spaces
cplnvar    -- strnsp, char(80) varying
ldspaces   -- number of leading spaces on input line
key        -- current input word to be investigated
```

ACTIVITY-COORDINATOR MODULE

```
DICTIONARY  -- virtual information dictionary
ENTITY      -- entity set representation
TOKEN       -- token representation
MACH        -- finite state machine representation
XTREE       -- execution tree representation
XCHNGE      -- entity set table representation

UNIT        -- current data base statement to be processed
TKLSPTR     -- pointer to the list of input tokens
GO          -- indicator to proceed with beyond the
                   tokenizer-PROCESSOR stage
```

TOKENIZER-PROCESSOR MODULE:

```
unit      -- current data base statement
tklsptr    -- pointer to list of input tokens
diction    -- dictionary
go        -- indicator to continue processing
             (set only for retrieval statements)
kind      -- numeric indicator for arithmatic and
             string constants.
word      -- first word of data base statement
```

5.0.0 LANGUAGE ILLUSTRATION AND SPECIFICATION

This section contains an illustration and a formal specification of the data base language implemented on the virtual information facility, as well as the buffer commands which are implemented to provide an interactive environment in which virtual information processing may be continued.

5.1.0 DATA BASE STATEMENTS

5.1.1 DEFINE STATEMENTS

A user may define the character string x to be a macro definition of the character string y by the following statement:

```
define x as y ;
def x as y ;
```

For instance:

```
define currentyear as birthyear + age ;
define old as age > 60 ;
define employee as {worker} ;
def a as 2+3+4/5*8 ;
def inches as 2.54 * centimeters ;
```

Using define statements to simulate functions:

```
define sum#a#b as #a + #b ;
```

This specifies a function with two arguments, #a and #b. The value of sum#2#4 when evaluated would be 6.

To remove previously defined definitions:

```
define age remove ;
define age rem ;
define a remove ;
```

5.1.2 ADHOC STATEMENTS

Adhoc statements are similar to defines statements; the only difference lie in the target catalogue identity. Adhoc statements operate on the adhoc dictionary, and define statements operate on the permanent dictionary.

```
adhoc x as y ;
adhoc curentyear as 1982 ;
adhoc sgfhg as kruilko ;
adhoc age rem ;
adhoc avg#x#y#z as (#x + #y + #z) / 3 ;
```

In both define and adhoc statements, virtual definition may be defined on top of other virtual definitions; in our implementation, we allow a maximum of 10 nested levels of virtual information definition. Thus, if one defines a recursive definition, our system would terminate the entire process of replacing definition names by their associated definitions by the eleventh attempt in replacing

the same definition name.

5.1.3 LISTDEF STATEMENTS

These statements list the stored definitions in the dictionary by name; the search order is:
adhoc --> permanent.

```
listdef age ;
listdef employee ;
listdef sgfhg ;
```

5.1.4 RETRIEVE STATEMENTS

Our retrieve statements are powerful enough to retrieve the following kinds of virtual information from either real or virtual entity sets:

a) computed facts
b) implied facts
c) factored facts

Computed facts are those information derived from an algorithmic computation on existing data; implied facts are those information derived from indirect associations; factored facts are those instances of a particular group of facts which share a certain condition on their attribute values.

We support the following kinds of computational

operators with four levels of precedence, left to right within each level of precedence, and together with parenthesized precedence capability:

+ , - , | ,	---	lowest order of precedence (plus, minus, and concatenate) (binary infix operators)
* , / ,	---	next order of precedence (multiplication and division) (binary infix operators)
!	---	next order of precedence (exponentiation operator)
+ , - ,	---	highest order of precedence (arithmatic pre-operators)

Aside from these built-in operators, we also support a number of built-in functions as enumerated below:

functions with no arguments:

date usage --> nextdate = STR (DATE , 2:'$') + 1 ; first, one would use the STRING function to obtain the relevant portion of the value returned by the DATE function, and then this value is incremented by 1.
A date in the system may be stored in the form month$date$year, or any other pre-determined

manner. The STRING function is very much suited for the getting of relevant portions of data stored in this form.

Functions with one argument:

These functions operate on entire entity sets; in this sense, they are vertical operators, not of the unilateral kind which we are usually familiar with.

Any valid expression may serve as an argument to built-in functions.

MAX(y) usage --> max (length + width + hight)
refers to that particular instance of the entity set whose dimensions have a greater sum than all other members of the set.

```
retrieve ( { employee} where ( salary = max (salary) ) )
        by ( {v0} name ) ;
```

gets the name of the employee who earns the highest salary.

SUM(y) usage --> where (sum (y) = 100)
yields true if the sum of all instances of y in the current entity set equals 100.

MIN(a+b-5)

for each member of the set, the value of argument expression is first calculated,

then the minimum of them all is taken.

ABS(x+y+z)

returns the absolute value of the argument expression for each member of the entity set.

SGN(index) -- returns -1 if argument is negative
returns 0 if argument is zero
returns 1 if argument is positive

SUM(x!2)

sums up the squares of the variable x, yielding one single value.

POS(v) -- returns boolean value for each instance of attribute v in the entity set.

ZER(x) -- returns boolean value for each instance of attribute x in the entity set.

Functions of more than one argument:

STR (b ,nth occurrence of 'x', mth occurrence of 'y')
returns a substring of b from the nth occurrence of 'x'
to the mth occurence of 'y' exclusively.
usage --> STR (b , 4:'x' , 5:'y')

a retrieve statement is of the following basic form:

```
retrieve ( list of real and/or virtual entity sets
           separated by commas and each with an optional
           predicate clause )
         by ( entity set designation
              list of items to be retrieved ) ;
```

The first set in the list would be known as {v0}
The second set in the list would be known as {v1}
" " " " " " " " " "
The tenth set in the list would be known as {v9}
A maximum of ten such sets on this level is permitted.

We hope to demonstrate the functionality of the retrieve statement through the following examples:

```
retrieve ( {es1} ) by ({V0} x) ;
```

gets all those "x" attributes of entity set "es1" .

```
retrieve ( {es1} ) by ( {v0} x+3 ) ;
```

computes and returns all x+3 instances of entity set ES1 which has the attribute X.

```
retrieve ( {es1},{es2} ) by ( {v0} max (x) )
                         by ( {v1} y*4, min (y*z) ) ;
```

First, it gets the instance of "es1" 's attribute "x" which has the highest value of all instances of "es1" 's attribute "x",

Second, it gets the "y*4" elements of entity set "es2", y being an attribute of "es2" , then it gets the instance of "es2" 's "y*z" which has the minimal value of all other instances of "es2" 's "y*z", "y" and "z", both being attributes of "es2" .

```
retrieve ( {es1} where ( ( ( x1 < x2 ) and ( x3 = x4 ) ) ) ,
           {es2} where ( y1 = ( 1,2,3,4,5 ) or y2 = y3 )    ,
           {{v0}} where ( x1 | x3 = str ( b,4:'$',5:'$' )))
         by ( {v0} x1, x3)
         by ( {v1} y1, y2) ;
```

A complete set of predicate conditions on real and virtual entity sets is supported with "and" , "or" , "xor" and "¬" connectors, with "<" , ">" , "=" , "¬=", "¬<" , and "¬>" relators. the default order of precedence is from left to right unless otherwise indicated by the use of parentheses.

For instance: the following are equivalent conditions:

```
x1 = x2 and x2 > x3 or x3 < x4
(x1 = x2) and (x2 > x3) or (x3 < x4)
((x1 = x2) and (x2 > x3)) or x3 < x4
(((((x1 = x2))))) and x2 > x3 or (x3 < x4)
```

Each where clause is attached to the entity set specified immediately prior to the clause itself; the

only restriction on the kinds of entity sets allowed to have where clauses attached to them is that they are not one of the following:

{v0},{v1},{v2},{v3},{v4}

{v5},{v6},{v7},{v8},{v9}

This is so because these entity sets only refer to some other entity set which was already specified. According to this principle, the following entity sets may have associated predicates because they are themselves the specification of new virtual entity sets:

{{v0}},{{v1}},

.....{{v9}}

The second entity set in the foregoing retrieve statement has an associated predicate which specified y1 = (1,2,3,4,5); this predicate requires y1 to be of either one of the constants within the enclosing set of parenthesis. however, when using this kind of comparison, we make the restriction that all values which appear in the enclosing set of parentheses must be either an arithmatic constant or a string constant.

The third condition clause in the foregoing retrieve statement illustrates the use of the string functions; the function call is attempting to return

the substring of b, from the 4th occurence of the '$' character to the 5th occurence of the '$' character. The predicate would yield true if the results of concatenating x1 and x3 is equal to the retrun value of that function call.

```
retrieve ( { {es1} mi (x,y,z) {es2} }
        by ( {v0} weight ) ;
```

This statement retrieves all instances of the weight attribute of the virtual entity set composed of the "multiple union" of real entity sets es1 and es2, based on the common attributes "x", "y", and "z".

Each virtual entity set, enclosed by a set of left and right braces, may itself be composed of two other virtual entity sets as the result of a set operation, and each of these two component entity sets may also be composed of two other virtual entity sets as the result of a set operation, and each of these component entity sets so on. In this manner, virtual entity sets may be built very quickly one on top of another, each with its own set of predicate conditions to be met.

Five set operators are supported between two entity sets: they are, multiple union, multiple intersection, single union, single intersection, and cartesian product; namely, MU, MI, SU, SI, and CS. The semantics of these

operators are described in the co-thesis by Peter Lu.

usage --> {{es1} SI (x) {es2}}

{{es1} su (y,z) {es2}}

{{es1} MU (y,z,z1) {es2}}

The operands of MU, MI, SU, and SI can be a list of attribute names separated by commas, but the operands to CS must be two in number and the first one must be preceded by a " " sign to indicate its cartesianess.

{{es1} cs (id,class) {es3}}

An arbitrary WHERE clause representing a predicate condition may follow each and every kind of prescribed virtual entity set.

{{es1} where (color = 'red') cs (id,class)
{es3} where (num > 7) } where (size < 5)

5.2.0 BUFFER COMMANDS

These interactive commands may be issued by the user via a terminal session with the virtual information facility. They are the means by which an interactive environment is constructed in which the data base commands may be executed.

The buffer is divided into an execution and a transaction buffer. an adhoc dictionary is built for the duration of each transaction in which many data base statements may be strung together and executed sequentially. Thus,

within a transaction a user may operate on either the permanent dictionary shared by itself and any other transaction executed before or after it, or the adhoc dictionary which is for its own exclusive use.

A completed data base statement in the execution buffer will automatically trigger the execution of that statement; therefore, the execution buffer is not suited for the stringing together of multiple statements.

Each buffer command may be entered from within either the transaction buffer or the execution buffer, and may be recognized by two or more initial characters of its full name. furthermore, the contens of the execution buffer and at least 10 lines of the transaction buffer will always be displayed on the terminal.

5.2.1 COMMAND SYNTAX

(1) FINPUT 1starg

This command will read the contents of the cms file whose file name is "file", and file type is whatever is entered as "1starg", into the transaction buffer. The original content of the transaction buffer before the execution of this command will be erased.

(2) FSAVE 1starg

This command will write into the cms file whose file name is "file", and file type is whatever is entered as "lstarg", from the contents of the transaction buffer. Upon completion of the command, transaction buffer content will be empty.

(3) TRANSACT

This command lets the user enter the transaction buffer.

(4) ENDTRANS

This command lets the user terminate the transaction buffer and enter the execution buffer.

(5) TERMINATE

This command terminates the virtual information facility and returns control to cms.

(6) RUNTRANS

This command executes the contents of the transaction buffer statement by statement.

(7) DODELETE

This command does the same as "runtrans" except that upon its completion, the transaction buffer contents will be erased.

(8) ERASETRANS

This command erases the contents of the transaction buffer.

(9) KILLEXEC

This command erases the contents of the execution buffer.

(10) HELP

This command gives a brief description of all buffer commands.

(11) INSERT 1starg 2ndarg

This command would insert a line of text into the transaction buffer. the first argument is a destination of the line number within the buffer after which the inserted line is to be inserted, and the second argument is the text to be inserted.

(12) DELETE 1starg

This command deletes a line from the current transaction buffer, and the exact line number is specified by the first argument.

(13) TOPLINE 1starg

This command specifies the starting line number of the ten transaction buffer lines which are always displayed, and that number is designated by the first argument.

5.3.0 FORMAL BNF DESCRIPTION OF DATA BASE STATEMENTS

****

<defstmt>::= "DEFINE" | "DEF" name defopt ;

```
<defopt> ::= < "AS" <***>  > |
             < "REMOVE" | "REM" >
<adhocstmt>   ::=  "ADHOC" name defopt ;
<liststmt>    ::=  "LISTDEF" name ;
*********************************************************************
<retrstmt>    ::=  "RETRIEVE" ( vsets ) byspec ;
<vsets>  ::=  < "{" vsets1 "}" { "where" ( cond ) } > |
              < "{" vindrs "}" >
              { , vsets }
<vsets1> ::=  name |
              combsets
<combsets>    ::=  < "{" vsets1 "}" { "where" ( cond ) } > |
              < "{" vindrs "}" >
              { setop vsets2 }
<vsets2> ::=  < "{" vsets1 "}" { "where" ( cond ) } > |
              < "{" vindrs "}" >

<vindrs> ::=  < "V0" | "V1" | "V2" | "V3" | "V4" | "V5"
                     | "V6" | "V7" | "V8" | "V9" >
              | < "{" vindrs "}" >

<setop>  ::=  <cs> | <ncs>
<ncs>    ::=  < "MI" | "SI" | "MU" | "SU" >
              ( reflist )

<reflist>::=  refl { , refl }
<cs>     ::=  "CS" (   varef , varef )
*********************************************************************
```

```
<exp>     ::=  <(exp)> | exp-infl | exp-inf2 | exp-pre
                       | exp-pwr  | exp-prim

<exp-infl>     ::=  exp infl-op <(exp)> | exp-inf2 | exp-prim
                                       | exp-pwr
<exp-inf2>     ::=  exp2 inf2-op <(exp)> | exp-prim | exp=pwr
<exp2>    ::=  <(exp2)> | exp-inf2 | exp-pre
                        | exp-pwr  | exp-prim
<infl-op>::=  + | - | "|"
<inf2-op>::=  * | /
<exp-pre>::=  pre-op < exp-prim | exp-pre | exp-pwr >
<pre-op>  ::=  + | -
<exp-prim>     ::=  ref | const
<exp-pwr>::=  exp-prim ! < exp-prim | exp-pre >
<ref>     ::=  refl | funref
<refl>    ::=  {   } varef
<varef>   ::=  name { ( varef ) }
<const>   ::=  fixed | integer
<fixed>   ::=  integer  . { integer }
<digit>   ::=  0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
<integer>::=  digit | digit integer
*****************************************************************
<funref>  ::=  singfun | strfun
<singfun>::=  funame ( exp )
<funame>  ::=  "MAX" | "MIN" | "ABS" | "POS"
                     | "SGN" | "ZER" | "SUM"

<strfun>  ::=  "str" ( strarg )
```

```
<strarg> ::=  exp , exp { : exp } strargl
<strargl>::=  { < @ exp { : exp } > | < , exp { : exp }

*********************************************************************

<cond>   ::=  < ( cond ) > | condl
<condl>  ::=  smpcnd | < cond condop cond2 >
<cond2>  ::=  smpcnd | < ( cond ) >
<smpcnd> ::=  < ( smpcnd ) > | < { not } ( smpcnd ) >
                             | < exp relop exp >
<not>    ::=  ¬ | <> not
<condop> ::=  "AND" | "OR" | "XOR"
<relop>  ::=  = | > | <
                | { not } =
                | { not } >
                | { not } <

*********************************************************************
name          any character string of length less than 17
              and composed only of the following characters:
              abcdefghijklmnopqrstuvwxyz$#&_¢?"
*********************************************************************
```

5.3.1 BNF SUPPLEMENT

* comments are enclosed within "\" characters
* quote characters within string constants are represented by two consecutive single quote characters
* string constants are enclosed by "'", single quote

```
    characters
*   <***> designates any arbitrary character string
*   single line comments enclosed by "\" characters are permitted
    before, after, and within each data base statement,
    as well as before and after each buffer command line.
    They are eventually removed, and are not recognized
    as part of any input line.
*   the system makes no distinction between lower and upper
    case characters.
*****************************************************************
```

6.0.0 FINITE-STATE-MACHINE (PUSH-DOWN-AUTOMATON)

In this chapter, the Finite-State-Machine used to parse data base retrieval statements would be briefly described. A Finite-State-Machine consists of a number of states, one of which is a start state, and one or many of which may be an ending state. There also is a pointer which would point to the current word being examined on the user given statement being processed. In our case, the user input is always a retrieval statement written in the data base language presented in Chapter5, and each word would always be a single token along the token chain to which the original retrieval statement has already been transformed to.

Each state within the machine has a set of match-next_state rules, and collectively the union of these sets of rules regulate the behavior of the finite-state machine on any given input. Each state attempts to find a match between the current word and the match section of any one of its rules, and if a match is found, then control is passed to the state identified by the next_state section of the matching rule, and the input pointer points to the next word of the statement being processed. If the end of input is ever reached, and control happens to be within an ending state, then the machine halts and is said to have accepted the statement which it had just processed.

Our construction of such a finite-state-machine went straight on to meet a number of problems. First, such a machine has no provisions for any processing except for moving from state to state. Thus, we augmented our design to a Push-Down-Automaton which is a finite-state-machine with auxiliary memory and data movement capabilities. In fact, in order to generate an execution tree and various tables from a given retrieval statement, we had to augment the processing ability of the automaton by a set of action routines, and transform the format of state-rules to a tri-tuple consisting of a match section, an action section, and a next_state section.

The auxiliary memory we have chosen for the automaton is in the form of two stacks, an operator stack and an operand stack. The match section of each rule has provisions to match either the current word on three different sources, the input token chain, the top of stack #1, and the top of stack #2. Actually, when the source is the input token chain, an added ability to match for a given class of tokens as well as a specific token is available. The action section of each rule has provisions for pushing and poping the current input token, onto or off either stack #1 or stack #2, and for invoking other action routines which generates an execution tree and various tables from the current elements on both stacks, and modifies the current contents of the stacks. The next_state section of each rule contains the state ID number which identifies the next state to

which control would be passed to after the proper action routines in the matching rule have been executed. Our machine is deterministic in nature; by this we mean that for a given combination of input token, top of stack #1, and stack #2, there is at most one next_state from which control may go to after the current state. A non-deterministic machine would have been condensed, but also more complex and harder to implement because of the need to backtrack over decision points.

The construction of this push-down-automaton is logically divided into two parts, the writing of the match-action-next_state rules, and the writing of a program which takes these rules as an input and sets up the proper environment in which data would be matched, actions would be performed, and next_states would be go to, exactly according to the specifications of the prescribed match-action-next_state rules. The front-end of the virtual information facility, as presented in this thesis, is responsible for the writing of these match-action-next-state rules, and the implementation of these rules is a responsibility of the back-end.

6.1.0 CONFIGURATION

Action routine implementations are part of the back-end written by Peter Lu in his concurrent thesis. These programs are within the PARSE module as illustrated in figure 4.1 . As

part of the front-end, the match-action-next_state rules are within the FINITE-STATE-MACHINE module and currently reside in CMS file "file machin" . A DEFMCH module is written to establish the machine environment, taking the contents of "file machin" as input, and the PARSE module, when called upon, activates the finite-state-machine.

6.2.0 MATCH-ACTION-NEXT_STATE RULES

Match-Action-Next_State rules is a 3-tuple of information. The first component presrcibes what to match for a certain element on either one or more than one of the following sourses, the input stream, top element of stack #1, and top element of stack #2. The second component is a sequence of action routine invocations; these routines are to be executed whenever the matching component of the same rule matches. The third component is a state-number representing the next state to which control is to go when the action routines in the same rule have been executed.

The three components of each rule is separated from each other by the "\" character as illustrated in the following:

\ match component \ action component \ next_state number \

Furthermore, since these rules prescribe the transitions from state to state, they are referred to as the "transiion rules", and the full specification of a transition rule is as follows:

t \ match component \ action component \ next_state number \

The specification of a state is accomplished first by writing the following to indicate the identity of the state:

s \ state number \

and then by a listing of the match-action-next_state rules which belong to this particular state. The sequential order of rules in this list can not be inter-changed, because when control comes to each state, the rules will be trid sequentially in the order of their position on the list. Thus, a sample state specification is as the following:

```
s \ 25 \
t \ retrieve \ del \ 26 \
t \ ( \ pop,1 \ 27 \
t \ + \ pop,2 \ 36 \
```

The foregoing rules would first try to match the word from the input, if it is matched, then the action routine "del",

delete input token, would be executed, and then control would go to state number 26. If the first rule did not match, then the second rule which attempts to match a "(" character on the input would be tried. If this rule matches, then the "pop" routine would be executed, and stack #1 would be popped, and control would go to state number 27. If the second rule did not match, then the machine would try the third rule, which matches for the "+" operator on the input stream, if it is matched, then stack #2 would be popped and control would go to state number 36. If none of the rules for a the current state matches, then the machine would signal premature termination, which means that the input is invalid, and diagnostic messages would be sent to CMS file "file error". State number 0 is the final state of the machine, and if control is ever passed to this state, then the input is valid, accepted, and successfully processed; the associated execution tree and entity set tables would have already been generated and available for use by the following stages of the virtual information facility.

The match component of each rule is composed of zero to three separate parts, each of which is separated from the other by a "," character. The first part represents the input source, the second part represents the source from stack #1, and the third part represents the source from stack #2. If non of the three parts exist, then that particular rule would match everything and anything, and the corresponding action routines would

always be executed if the machine ever tries to match that rule.

For instance:

```
s\1\
t\a,b,c\del\2\
```

The foregoing rule would match simultaneously a character "a" on the input stream, a character "b" from the top of stack #1, and a "c" character from the top of stack #2. All three sources must be matched before the corresponding action routines may be executed. If any of the sources does not match, then this entire rule is not matched, and either the next rule in the sequence would be matched or the machine would signal premature termination if there are no more rules to be matched for this state. Thus, at most three sources may be matched in the match component of the rule, and at most one single token may be matched on any one source.

The action component of each rule may contain calls to more than one action routine. These action routine invocations are written in sequential order and are separated by the "|" character; these routines would be executed in the order of there appearence in the action component. For instance:

```
s\5\
```

```
t\ + \ push,2,i@ | del | pop,1 \ 6 \
```

The foregoing rule would match the "+" character on the input stream, and then execute the three action routines in sequential order. First, it would push the "+" character on to stack #2, as specified by the first routine call, then it would delete the current character on the input stream, thereby advancing the input pointer to point to the next input token, and then it would pop stack #1, popping off stack #1's top element.

In order to facilitate the matching of a group of symbols, not necessarily all of the same classification, we have developed the concept of a "cluster"; a cluster simply is a union of one or more prescribed symbols which may be matched under one cluster name. All cluster names begin with a "@" character, and they provide an added convenience for the making of transition rules. For instance, an arithmatic cluster may include the + and - characters, and be named "@sumop". By using the name @sumop in the match component, we may match either the + or the - characters. We currently have the following groups of clusters:

```
@sumop    ---   + , -
@sumop>   ---   + , - , * , / , !
@multop   ---   * | /
```

```
@multop>  ---   * , / , !
@conc     ---   |
@conc>    ---   | , + , - , * , / , !
@virt     ---   v0 , v1 , v2 , v3 , v4 ,
                v5 , v6 , v7 , v8 , v9
@rel      ---   > , <
@cmp      ---   and , or , xor
@setop    ---   cs , mu , mi , su , si
```

It was mentioned earlier that a rule may match from the input source a token of a specific classification; to do this, the rule indicates the class of characters it would be matching for by writing a ":" character and followed by the class representation character. There are altogether eight different classes of tokens, namely the classes A , N , D , O , B , Q , M , and S .

Characters belonging to class A are the following:

abcdefghijklmnopqrstuvwxyz

Characters belonging to class N are the following:

0123456789

Characters belonging to class D are the following:

.

Characters belonging to class O are the following:

+-/*!|

Characters belonging to class B are the following:

¬=><,()°{}

Characters belonging to class Q are the following:

'

Characters belonging to class M are the following:

"?¢_&±#$

Characters belonging to class S are the following:

%

The following rule matches for a token of class M on the input stream, does nothing, and then passes control to state 4.

```
s \ 3 \
t \ :M \ \ 4 \
```

The parsing of any language may sometimes be facilitated by the creation of sub-pasers which parse a subset of the language. The usefulness of this idea is demonstrated by our using of the sub-routine concept in the finite-state-machine. Two sub-machines were written, one to parse expressions, and one to parse entity set conditions which calls on the

expression parser. The idea is to pass control to the sub-parser, and leave subroutine return command and address on top of stack #2 before entering the sub-parser. When the sub-parser finds a negative state number as the next_state, it should have that return command and address available on top of stack #2, and should pop that element off stack #2, and then pass control to the state identified by that command. For example, the following rule passes control to a sub-parser which starts on state 30, and also specifies the return state number as 80 when the sub-parser finds a negative-state number in the next_state component. With this strategy, the last rule which indicate the successful parse of the sub-parser must have a negative state number in the next_state component.

```
s \ 20 \
t \ d \ del | push,2,subr:80 \ 30 \
```

6.3.0 ACTION ROUTINES

The following action routines are written within the back-end of the facility and are available for use in the action component of the match-action-next_state rules:

Routine	Usage	Semantics

POP	pop,1	pops stack #1
	pop,2	pops stack #2
PUSH or P	push,1,i@	pushes the input
	p,1,i@	token onto stack #1
	push,1,c@	pushes the input token concatenated by a ":" and its classification onto stack #1
	push,2,i@:2	pushes the input token concatenated by ":2" onto stack #2. Frequently, this is used to associate the expected number of operands to the operator which is being pushed onto the operator stack.
DEL	del	advances the input pointer to point to the next input token.

Non-stack related routines:

GENNODE or GD

generates an operator node with its specified number of children which are found on stack #1 as a partial execution tree. The address to the partial tree just generated is placed back on top of stack #1.

INDX

adds the first level of indirection to a data element

ADDON

adds an additional level of indirection

VIRTX

exchanges the addresss of the indicated virtual entity set

MULX

generates a multiple "OR" node

ATTWHR

attaches the address of a condition to its associated virtual entity set

GENENT

generates a new virtual entity set

VIRTA

adds the cuurent virtual entity set to an entity set table

The non-stack relate routines listed above have much to do with the internal workings of the back-end and are left to be more precisely explained by the back-end documentation.

6.4.0 LISTING

The rules written in CMS file "file machin" are not readily readable because of its syntax; a FORMMCH program is available to format it to a readable form as shown in a listing of the rules on the following pages:

FINITE-STAE-MACHINE RULES
MATCH - ACTION - NEXT_STATE

```
STATE NUMBER:    1
   1.  1  RETRIEVE                  PUSH,2,I@¦DEL¦PUSH,2,SUBR:2          47

STATE NUMBER:    2
   2.  1  ;,,RETRIEVE                                                     0
   2.  2  BY                        DEL¦PUSH,2,SUBR:2                    140

STATE NUMBER:    3

STATE NUMBER:   11
  11.  1  -                         DEL                                  18
  11.  2  (                         PUSH,2,I@¦DEL                        12
  11.  3  :A¦:S                     PUSH,1,C@¦DEL                        12
  11.  4  -                         PUSH,2,I@P:1¦DEL                     16
  11.  5  +                         DEL                                  11
  11.  6  :R                        PUSH,1,C@¦DEL                        14
  11.  7  STR                       PUSH,2,SUBR:12¦PUSH,2,I@:6¦DEL       22
  11.  8  DATE                      PUSH,2,I@:O¦GENNODE¦DEL              12
  11.  9  :B                        PUSH,2,SUBR:12¦PUSH,2,I@:1¦DEL       20

STATE NUMBER:   12
  12.  1  )..(                      DEL¦POP,2                            12
  12.  2  ,,-P                      GENNODE                              12
  12.  3  !                         PUSH,2,I@:2¦DEL                      13
  12.  4  @SUMOP,,@SUMOP>           GENNODE                              12
  12.  5  @SUMOP>,,@CONC            PUSH,2,I@:2¦DEL                      13
  12.  6  @MULTOP,,@MULTOP>         GENNODE                              12
  12.  7  @MULTOP,,@SUMOP           PUSH,2,I@:2¦DEL                      13
  12.  8  @CONC,,@CONC>             GENNODE                              12
  12.  9  @CONC>                    PUSH,2,I@:2¦DEL                      13
  12. 10  ,,SUBR                                                         -1
  12. 11                            GENNODE                              12

STATE NUMBER:   13
  13.  1  :R                        PUSH,1,C@¦DEL                        14
  13.  2  :A¦:S                     PUSH,1,C@¦DEL                        12
  13.  3  -                         PUSH,2,I@P:1¦DEL                     16
  13.  4  +                         DEL                                  13
  13.  5  (                         PUSH,2,I@¦DEL                        13
  13.  6  :B                        PUSH,2,I@:1¦DEL                      17

STATE NUMBER:   14
  14.  1  (                         PUSH,2,I@¦DEL¦INDX,1                 15
  14.  2                                                                 12

STATE NUMBER:   15
  15.  1  :R                        PUSH,1,I@¦DEL                        19

STATE NUMBER:   16
  16.  1  -                         PUSH,2,I@P:1¦DEL                     16
  16.  2  +                         DEL                                  16
  16.  3  :A¦:S                     PUSH,1,C@¦DEL                        12
  16.  4  :R                        PUSH,1,C@¦DEL                        14
```

FINITE-STATE-MACHINE RULES
MATCH - ACTION - NEXT_STATE

```
STATE NUMBER:  17
  17. 1  (                    PUSH,2,I@¦DEL                                  11

STATE NUMBER:  18
  18. 1  :R                   PUSH,1,C@;;C¦DEL                               14

STATE NUMBER:  19
  19. 1  (                    PUSH,2,I@¦DEL¦ADDON,1@¦POP,1                   15
  19. 2  ),,(                 ADDON,1@¦POP,1¦DEL¦POP,2                       28

STATE NUMBER:  20
  20. 1  (                    DEL¦PUSH,2,SUBR:21                             11

STATE NUMBER:  21
  21. 1  ,,SUBR                                                              -1
  21. 2  )                    DEL¦GENNODE                                    21

STATE NUMBER:  22
  22. 1  (                    DEL¦PUSH,2,SUBR:23                             11

STATE NUMBER:  23
  23. 1  ,,SUBR                                                              -1
  23. 2  %O                   DEL¦PUSH,2,SUBR:24                             11

STATE NUMBER:  24
  24. 1  ,,SUBR                                                              -1
  24. 2  )                    DEL¦P,1,:-1¦P,1,:-1¦P,1,:-1¦P,1,:-1¦GD         24
  24. 3  %O                   DEL¦P,1,:-1¦P,1,:-2¦P,2,SUBR:26                11
  24. 4  %5                   DEL¦P,1,:-1¦P,1,:-3¦P,2,SUBR:26                11
  24. 5  %3                   DEL¦P,2,SUBR:25                                11

STATE NUMBER:  25
  25. 1  ,,SUBR                                                              -1
  25. 2  )                    DEL¦P,1,:-1¦P,1,:-1¦P,1,:-1¦GD                 25
  25. 3  %O                   DEL¦P,1,:-2¦PUSH,2,SUBR:26                     11
  25. 4  %5                   DEL¦P,1,:-3¦PUSH,2,SUBR:26                     11

STATE NUMBER:  26
  26. 1  ,,SUBR                                                              -1
  26. 2  )                    DEL¦P,1,:-1¦GENNODE                            26
  26. 3  %3                   DEL¦PUSH,2,SUBR:27                             11

STATE NUMBER:  27
  27. 1  ,,SUBR                                                              -1
  27. 2  )                    DEL¦GENNODE                                    27

STATE NUMBER:  28
  28. 1  ),,(                 DEL¦POP,2                                      28
  28. 2                                                                      12

STATE NUMBER:  29
```

FINITE-STATE-MACHINE RULES
MATCH - ACTION - NEXT_STATE

```
STATE NUMBER:  30
  30. 1  (                    PUSH,2,I@¦DEL                    31

STATE NUMBER:  31
  31. 1  ^                    PUSH,2,I@:1¦DEL                  36
  31. 2  (                    PUSH,2,I@¦DEL                    32
  31. 3                       PUSH,2,SUBR:33                   11

STATE NUMBER:  32
  32. 1  (                    PUSH,2,I@¦DEL                    32
  32. 2                       PUSH,2,SUBR:33                   11

STATE NUMBER:  33
  33. 1  ^                    PUSH,2,I@:1¦DEL                  37
  33. 2  ),,(                 POP,2¦DEL                        33
  33. 3  =                    PUSH,2,I@:2¦DEL                  38
  33. 4  @REL                 PUSH,2,I@:2¦DEL                  34

STATE NUMBER:  34
  34. 1  (                    PUSH,2,I@¦DEL                    34
  34. 2                       PUSH,2,SUBR:35                   11

STATE NUMBER:  35
  35. 1  ),,(                 POP,2¦DEL                        35
  35. 2  ,,@REL               GENNODE                          35
  35. 3  ,,^                  GENNODE                          35
  35. 4  ,,@CMP               GENNODE                          35
  35. 5  @CMP                 PUSH,2,I@:2¦DEL                  31
  35. 6  ,,SUBR                                                -1

STATE NUMBER:  36
  36. 1  ^                    PUSH,2,I@:1¦DEL                  36
  36. 2  (                    PUSH,2,I@¦DEL                    32

STATE NUMBER:  37
  37. 1  ^                    PUSH,2,I@:1¦DEL                  37
  37. 2  @REL                 PUSH,2,I@:2¦DEL                  34

STATE NUMBER:  38
  38. 1  (                    PUSH,2,I@¦DEL                    39
  38. 2                                                        34

STATE NUMBER:  39
  39. 1  :A¦:S                PUSH,1,I@¦DEL                    40
  39. 2                                                        34

STATE NUMBER:  40
  40. 1  ),,(                 DEL¦POP,2¦GENNODE                35
  40. 2  %0                   DEL¦MULX,1                       41
```

FINITE-STATE-MACHINE RULES
MATCH - ACTION - NEXT_STATE

STATE NUMBER: 41		
41. 1 :A¦:S	PUSH,1,I@¦DEL	42
STATE NUMBER: 42		
42. 1),,(	DEL¦POP,2¦ADDON,1@¦POP,1¦GENNODE	35
42. 2 %O	DEL¦ADDON,1@¦POP,1	41
STATE NUMBER: 43		
STATE NUMBER: 47		
47. 1 (	DEL	48
STATE NUMBER: 48		
48. 1 (	PUSH,2,I@¦DEL	54
STATE NUMBER: 49		
49. 1),@VIRT	VIRTX,1,1@¦DEL	52
49. 2)	GENENT¦DEL	50
STATE NUMBER: 50		
50. 1 WHERE	DEL¦PUSH,2,SUBR:51	30
50. 2		52
STATE NUMBER: 51		
51. 1	ATTWHR	52
STATE NUMBER: 52		
52. 1 ;		53
52. 2 BY		53
52. 3)		53
52. 4 %O	DEL	48
	DEL¦VIRTA¦PUSH,2,SUBR:52	
STATE NUMBER: 53		
53. 1 .,SUBR		-1
STATE NUMBER: 54		
54. 1 :R	PUSH,1,I@¦DEL	49
54. 2 @VIRT	PUSH,1,I@¦DEL	49
54. 3 (		100
STATE NUMBER: 55		
55. 1 (	PUSH,2,I@¦DEL	56
STATE NUMBER: 56		
56. 1 :R	PUSH,1,I@¦DEL	57
56. 2 @VIRT	PUSH,1,I@¦DEL	57
56. 3 (		112

```
                                FINITE-STATE-MACHINE RULES
                                MATCH - ACTION - NEXT_STATE

STATE NUMBER:  57
  57. 1  ),@VIRT                VIRTX,1,1@¦DEL                    60
  57. 2  )                      GENENT¦DEL                        58

STATE NUMBER:  58
  58. 1  WHERE                  DEL¦PUSH,2,SUBR:59                30
  58. 2                                                           60

STATE NUMBER:  59
  59. 1                         ATTWHR                            60

STATE NUMBER:  60
  60. 1  @SETOP                                                   70
  60. 2  ,,SUBR                                                  -1
  60. 3  )                      GENENT¦DEL                        58

STATE NUMBER:  61
  61. 1  (                      PUSH,2,I@¦DEL                     62

STATE NUMBER:  62
  62. 1  :R                     PUSH,1,C@¦DEL                     63
  62. 2  @VIRT                  PUSH,1,I@¦DEL                     63
  62. 3  (                                                        124

STATE NUMBER:  63
  63. 1  ),@VIRT                VIRTX,1,1@¦DEL                    65
  63. 2                         GENENT¦DEL                        64

STATE NUMBER:  64
  64. 1  WHERE                  DEL¦PUSH,2,SUBR:65                30
  64. 2                                                           66

STATE NUMBER:  65
  65. 1                         ATTWHR                            66

STATE NUMBER:  66
  66. 1  ,,SUBR                                                  -1
  66. 2  )                      GENENT¦DEL                        64

STATE NUMBER:  67
```

FINITE-STATE-MACHINE RULES
MATCH - ACTION - NEXT_STATE

STATE NUMBER: 70		
70. 1 CS	PUSH,2,I⊕¦DEL	71
70. 2		80
STATE NUMBER: 71		
71. 1 (	PUSH,2,I⊕¦DEL	72
STATE NUMBER: 72		
72. 1 ~	DEL	73
STATE NUMBER: 73		
73. 1 :R	PUSH,1,C⊕;V;C¦DEL	74
STATE NUMBER: 74		
74. 1 (	PUSH,2,IND(¦DEL¦INDX,1	75
74. 2		78
STATE NUMBER: 75		
75. 1 :R	PUSH,1,I⊕¦DEL	76
STATE NUMBER: 76		
76. 1 (	PUSH,2,IND(¦DEL¦ADDON,1⊕¦POP,1	75
76. 2).,IND(	ADDON,1⊕¦POP,1¦DEL¦POP,2	77
STATE NUMBER: 77		
77. 1).,IND(	DEL¦POP,2	77
77. 2		78
STATE NUMBER: 78		
78. 1 %O	PUSH,2,I⊕¦DEL	90
STATE NUMBER: 79		
STATE NUMBER: 80		
80. 1	PUSH,2,I⊕¦DEL	81
STATE NUMBER: 81		
81. 1 (	PUSH,2,I⊕¦DEL	82
STATE NUMBER: 82		
82. 1 :R	PUSH,1,C⊕;V¦DEL	83
STATE NUMBER: 83		
83. 1)	PUSH,2,I⊕¦DEL	61
83. 2 %O	PUSH,2,I⊕¦DEL	82
83. 3 (	PUSH,2,IND(¦DEL¦INDX,1	84
STATE NUMBER: 84		
84. 1 :R	PUSH,1,I⊕¦DEL	85

FINITE-STATE-MACHINE RULES
MATCH - ACTION - NEXT_STATE

```
STATE NUMBER:  85
  85. 1  (                    PUSH,2,IND(¦DEL¦ADDON,1@¦POP,1          86
  85. 2  ),,IND(              ADDON,1@¦POP,1¦DEL¦POP,2                86

STATE NUMBER:  86
  86. 1  ),,IND(              DEL¦POP,2                               86
  86. 2                                                               87

STATE NUMBER:  87
  87. 1  )                    PUSH,2,I@¦DEL                           61
  87. 2  %O                   PUSH,2,I@¦DEL                           82

STATE NUMBER:  88

STATE NUMBER:  90
  90. 1  :R                   PUSH,1,C@;V;¦DEL                        91

STATE NUMBER:  91
  91. 1  (                    PUSH,2,IND(¦DEL¦INDX,1                  92
  91. 2                                                               94

STATE NUMBER:  92
  92. 1  :R                   PUSH,1,C@;V;¦DEL                        93

STATE NUMBER:  93
  93. 1  (                    PUSH,2,IND(¦DEL¦ADDON,1@¦POP,1          92
  93. 2  ),,IND(              ADDON,1@¦POP,1¦DEL¦POP,2                94

STATE NUMBER:  94
  94. 1  ),,IND(              DEL¦POP,2                               94
  94. 2  )                    PUSH,2,I@¦DEL                           61

STATE NUMBER:  95

STATE NUMBER:  100
 100. 1  ,,(                  POP,2                                   101
 100. 2                       PUSH,2,SUBR:52                          111

STATE NUMBER:  101
 101. 1  ,,(                  POP,2                                   102
 101. 2                       PUSH,2,SUBR:52                          110

STATE NUMBER:  102
 102. 1  ,,(                  POP,2                                   103
 102. 2                       PUSH,2,SUBR:52                          109

STATE NUMBER:  103
 103. 1  ,,(                  POP,2                                   104
 103. 2                       PUSH,2,SUBR:52                          108
```

FINITE-STATE-MACHINE RULES
MATCH - ACTION - NEXT_STATE

STATE NUMBER: 104		
104. 1 ..{	POP,2	105
104. 2	PUSH,2,SUBR:52	107
STATE NUMBER: 105		
105. 1	PUSH,2,SUBR:52	106
STATE NUMBER: 106		
106. 1	PUSH,2,{	107
STATE NUMBER: 107		
107. 1	PUSH,2,{	108
STATE NUMBER: 108		
108. 1	PUSH,2,{	109
STATE NUMBER: 109		
109. 1	PUSH,2,{	110
STATE NUMBER: 110		
110. 1	PUSH,2,{	111
STATE NUMBER: 111		
111. 1		55
STATE NUMBER: 112		
112. 1 ..{	POP,2	113
112. 2	PUSH,2,SUBR:60	123
STATE NUMBER: 113		
113. 1 ..{	POP,2	114
113. 2	PUSH,2,SUBR:60	122
STATE NUMBER: 114		
114. 1 ..{	POP,2	115
114. 2	PUSH,2,SUBR:60	121

FINITE-STATE-MACHINE RULES
MATCH - ACTION - NEXT_STATE

STATE NUMBER: 115		
115. 1 ..(	POP,2	116
115. 2	PUSH,2,SUBR:60	120
STATE NUMBER: 116		
116. 1 ..(	POP,2	117
116. 2	PUSH,2,SUBR:60	119
STATE NUMBER: 117		
117. 1	PUSH,2,SUBR:60	118
STATE NUMBER: 118		
118. 1	PUSH,2,(	119
STATE NUMBER: 119		
119. 1	PUSH,2,(	120
STATE NUMBER: 120		
120. 1	PUSH,2,(	121
STATE NUMBER: 121		
121. 1	PUSH,2,(	122
STATE NUMBER: 122		
122. 1	PUSH,2,(	123
STATE NUMBER: 123		
123. 1		55
STATE NUMBER: 124		
124. 1 ..(	POP,2	125
124. 2	PUSH,2,SUBR:66	135
STATE NUMBER: 125		
125. 1 ..(	POP,2	126
125. 2	PUSH,2,SUBR:66	134
STATE NUMBER: 126		
126. 1 ..(	POP,2	127
126. 2	PUSH,2,SUBR:66	133
STATE NUMBER: 127		
127. 1 ..(	POP,2	128
127. 2	PUSH,2,SUBR:66	132
STATE NUMBER: 128		
128. 1 ..(	POP,2	129
128. 2	PUSH,2,SUBR:66	131

FINITE-STATE-MACHINE RULES
MATCH - ACTION - NEXT_STATE

STATE NUMBER: 129 129. 1	PUSH,2,SUBR:66	130
STATE NUMBER: 130 130. 1	PUSH,2,(	131
STATE NUMBER: 131 131. 1	PUSH,2,(	132
STATE NUMBER: 132 132. 1	PUSH,2,(	133
STATE NUMBER: 133 133. 1	PUSH,2,(	134
STATE NUMBER: 134 134. 1	PUSH,2,(	135
STATE NUMBER: 135 135. 1		55
STATE NUMBER: 136		
STATE NUMBER: 140 140. 1 (	DEL	141
STATE NUMBER: 141 141. 1 (	DEL	142
STATE NUMBER: 142 142. 1 @VIRT	PUSH,1,I@¦DEL	143
STATE NUMBER: 143 143. 1)	DEL¦BYENT¦POP,1	144
STATE NUMBER: 144 144. 1) 144. 2	DEL PUSH,2,SUBR:145	53 11
STATE NUMBER: 145 145. 1	ATTBY¦POP,1	146
STATE NUMBER: 146 146. 1) 146. 2 %0	DEL DEL¦PUSH,2,SUBR:145	53 11
STATE NUMBER: 147 MACHINE DEFINITION COMPLETE		

7.0.0 MAJOR DESIGN ISSUES

The following are some of the decisions which we had to make in the design of the virtual information facility.

7.1.0 FORM OF STORAGE FOR VIRTUAL DEFINITIONS

How can virtual definitions be stored? We had two viable alternatives. One way is to store the definitions just as they are, in the form of character strings, and when in use, the definition would be substituted within the actual data base retrieval statement in place of the virtual definition name. The alternative to this strategy is to parse the definitions ahead of time, generate the associated execution tree and entity set tables, and when in actual use, the partial execution tree would be simply attached to the main execution tree as an extended subtree, and the main entity set tables would simply be augmented to include the partial table built from the definitions.

The method of parsing the definitions first is similar to the process of compilation. When in actual use, previously defined definitions need not be processed again and again. The method of storing the definitions as they are is similar to the process of interpretation. Each time a definition is used, the

entire process of parsing, tree building and table generation would have to be repeated.

Storing definitions as they are enhances the flexibility of virtual information. Definitions may be created, modified, and even deleted with great ease and efficiency. Furthermore, it eliminates the need to rebuild itself when users request a listing of the stored definitions. It also would enable a generalized macro facility in which not only legitimate and coherent definitions may be stored, but also the seemingly illogical and incoherent definitions as well.

Parsing the definitions as soon as they are defined is not an easy task. Many times, without the proper context in which the definitions are to be used, the associated semantics are not always clear. Even if we can get around this problem by restricting the potential contexts in which each definition may be used, we still would be encounter complicated problems in frequent tree manipulation. Re-shaping an execution tree is a very "messy" task, and would be prone to erroneous branch connections; traversing a huge tree is also not a reasonably efficient operation.

Thus, mainly for the foregoing reasons, we have decided on the first strategy, storing them as they are until invocation, to store virtual definitions.

7.2.0 PARSER STRUCTURE

Two strategies were given serious consideration for the parsing of data base statements written in the language specified in Chapter5. One method is to construct a FINITE-STATE-MACHINE which includes a set of match-action-next_state rules that correspond to the grammar rules of the data base language. In this manner, these match-action-next_state rules are inputs to the actual parser just like the data base statements which are to be parsed; with this approach, changes in grammar rules readily correspond to changes in the machine rules. The other method is to construct a conventional parser in which grammar rules are part of the parser program itself.

The finite-state-machine strategy has a highly modular characteristic and gives added flexibility to the data base language in terms of modifiability; however, it would be the first of such machines ever written by the author. The decision was made in favor of the finite-state-machine because the definite gains of this approach seem to surpass the potential for failure of its implementation.

7.3.0 PROGRAM CONTROL STRUCTURE

A decision was made to implement a centralized and horizontal control structure for the passing of program control from one to another. The alternative is to build a vertical control structure in which modules are nested one within another, and control may propagate many levels deep before suddenly jumping out to the top. The centralized control structure features an activity coordinator to which control must return to from each module before it is passed to another. Although the vertical approach may seem more natural, the horizontal approach is more adapted to the idea of a single virtual information level within the hierarchical design of INFOPLEX. Furthermore, the horizontal approach contributes more to program modularity with its regard for each module as a separate and un-nested entity. For these reasons, the decision was made to build a centralized and horizontal control structure.

7.4.0 INTERACTIVE EDITOR

Consideration was given to the question of whether or not to build an interactive, full-screen editor in real-time, similar to a miniature EMACS or XEDIT editor as part of the user-interface developed for the virtual information facility. The seriousness of the consideration remained questionable to this day. The argument against it is that the buffer program already supports the capability of inputing the transaction buffer content from an arbitrary CMS file; a user of virtual

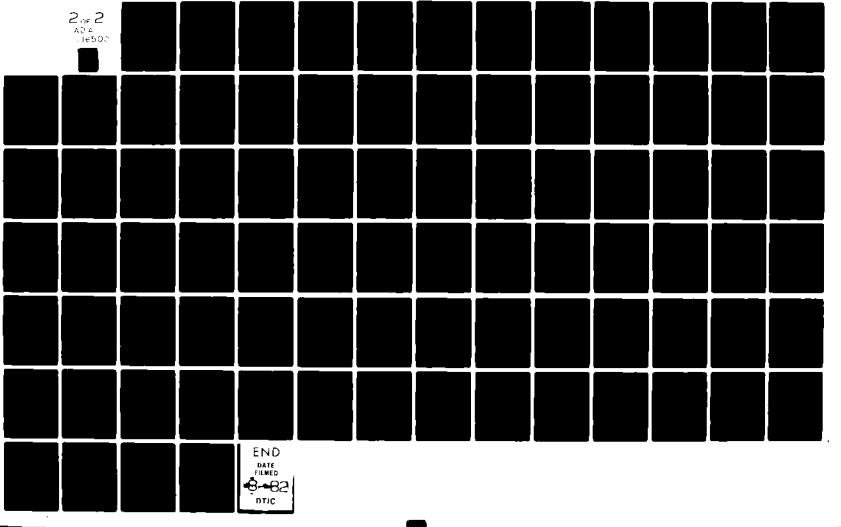
information may readily use the XEDIT editor available on CMS to edit their transaction stored in a CMS file, and later input that transaction to the transaction buffer through the FINPUT buffer command. Our interactive, full-screen line editor would take some effort to develop, and still would not be nearly as powerful as XEDIT. In other words, resources may be better utilized if spent on other areas of the virtual information facility. The argument for such an editor is simply that it would provide the added flexibility to change modify buffer contents from within the virtual information facility.

Finally, a decision was made to build a primitive line editor with display capabilities. This is a compromise between the two extremes; not much resources in terms of man-hours would be spent building such an editor, and it would give users of virtual information an added flexibility and convenience in being able to edit their transaction buffer content from within the facility.

7.5.0 LANGUAGE DESIGN

A decision was made to support infix arithmatic and string operators instead of operators in prefix or lisp notation. Although infix operators give rise to a language more difficult to parse, they are more user-friendly. Also, a decision was made to support the capability of explicitly over-riding the

AD-A116 902 ALFRED P SLOAN SCHOOL OF MANAGEMENT CAMBRIDGE MA F/G 9/2
VIRTUAL INFORMATION FACILITY OF THE INFOPLEX SOFTWARE TEST VEHI--ETC(U)
MAY 82 J LEE N00039-81-C-0663
UNCLASSIFIED M010-8205-10 NL



natural operator precedences by the use of parentheses in arithmatic, string, as well as boolean expressions. This capability makes more difficult the parsing process, but gives much added power and flexibility to the language. In essence, the added advantages of infix operators and use of parentheses for specification of precedence are considered well worth their cost of implementation.

8.0.0 CONCLUSION

Progress was made steadily and swiftly all through the first two months of design and implementation. Then, as precious time passed by each day, increasing hours of work were required for the prompt completion of all thesis objectives. Eventually, all available time was devoted to thesis work and efforts on academic courses became nearly non-existent.

Finally, the complete design and an initial version of the implementation were completed and integrated with Peter Lu's back-end to set up the first virtual information facility in operation on the INFOPLEX software test vehicle, a software simulation of the INFOPLEX data base computer. An extensive set of improvised test cases were written and tested on the facility. The internal interface to the back-end, namely, the instructions issued within the finite-state rules does to conform to expectations. Although the back-end is not yet able to integrate with the lower level of INFOPLEX to access real data, it is able to generate correct information requests based on the execution tree and entity set table established through finite-state-machine instructions which are issued within the front-end.

The results of the implementation give support to the design decisions which were made, especially the decision to con-

struct a finite-state machine and to keep virtual definitions as they are. Most thesis objectives were achieved except for the need of more rigorous test cases to establish the integrity of the facility. Instantly, this facility, when eventually integrated to the next level of INFOPLEX hierarchy, would greatly extend the power and capability of the data base.

Bibliography

1. Harry R. Lewis, Christos H. Papadimitriou,
   'Elements of the Theory of Computation'
   Prentice-Hall Software Series

2. David Gries,
   'Compiler Construction for Digital Computers'
   John Wily & Sons

3. Jeffrey Folinus, Stuart E. Madnick, Howard Schutzman,
   'Virtual Information in Data Base Computers'
   Center for Information Systems Research, M.I.T.

4. A Klug, D Tsichritzis,
   Multiple View Support Within the Ansi/Sparc Framework
   Center for Information Systems Research, M.I.T.

5. Meichun Hsu
   'FSTV - The Software Test Vehicle for the Functional
   Hierarchy of the INFOPLEX Data Base Computer'
   Center for Information Systems Research, M.I.T.

6. Thomas A. Standish
   'Data Structure Techniques'
   Addison-Wesley Computer Science Series

7. Aho Ullman
   'Principles of Compiler Design'
   Addison-Wesley Computer Science Series

8. Tak To
   'SHELL: A Simulation for the Software Test Vehicle
   of the INFOPLEX Data Base Computer'
   Center for Information Systems Research, M.I.T.

9. Chat-Yu Lam, Stuart E. Madnick
   'INFOPLEX Data Base Computer Architecture'
   Center for Information Systems Research, M.I.T.

10. Bruce Blumberg
    'INFOSAM - A Sample Database Management System'
    Center for Information Systems Research, M.I.T.

APPENDIX

```
                                                                        USI00010
                                                                        USI00020
                                                                        USI00030
                                                                        USI00040
                                                                        USI00050
USINT: PROC OPTIONS (MAIN);                                             USI00060
%INCLUDE DICTION;                                                       USI00070
%INCLUDE TOKEN;                                                         USI00080
DCL (NEWBUF, ACTCRD) ENTRY EXTERNAL;                                    USI00090
DCL DCTNRY ENTRY (,CHAR (200) VAR)                                      USI00100
          EXTERNAL RETURNS (CHAR (160) VAR);                            USI00110
DCL (LINE,GARBAGE) CHAR (80) ;                                          USI00120
DCL FIRSTLAST BIT(1) INIT ('0'B);                                       USI00130
DCL (EXECBUFF,TRNSBUFF) CHAR (4000) VAR INIT ('');                      USI00140
DCL PLIST1(3) CHAR (8) INIT ('DVHUTL','TEST',HIGH(8)),                  USI00150
    PLIST2(3) CHAR (8) INIT ('DVHUTL','CLEAR',HIGH(8)),                 USI00160
    (HIGH,PLIRETC) BUILTIN,                                             USI00170
    RETCODE FIXED BINARY (31,0),                                        USI00180
    CMSCMD EXTERNAL ENTRY OPTIONS (ASSEMBLER INTER);                    USI00190
DCL JUNK CHAR (160) VAR;                                                USI00200
                                                                        USI00210
JUNK = DCTNRY (DICTION,'INITIALIZE');                                   USI00220
CALL CMSCMD (PLIST2,RETCODE);                                           USI00230
PUT SKIP EDIT ('***INFOPLEX DATA BASE MACHIN***')                       USI00240
             (COL(15),A);                                               USI00250
PUT SKIP EDIT ('TYPE "VIR" FOR VIRTUAL INFORMATION PROCESSOR')          USI00260
             (COL(1),A);                                                USI00270
PUT SKIP EDIT ('TYPE "REAL" FOR REAL INFORMATION PROCESSOR')            USI00280
             (COL(1),A);                                                USI00290
GET EDIT (LINE) (COL(1),A(80));                                         USI00300
LINE = TRANSLATE (LINE, 'ABCDEFGHIJKLMNOPQRSTUVWXYZ',                  USI00310
                        'abcdefghijklmnopqrstuvwxyz');                  USI00320
IF INDEX (LINE,'REAL') ^= 0 THEN RETURN;                                USI00330
ELSE IF INDEX (LINE, 'VIR') = 0 THEN DO;                                USI00340
   PUT EDIT ('UNRECOGNIZABLE SUBSPACE NAME, PROGRAM TERMINATED!')       USI00350
            (COL(1),A(80));                                             USI00360
   RETURN;                                                              USI00370
   END;                                                                 USI00380
FIRSTLAST = '1'B;                                                       USI00390
DO WHILE ('1'B);                                                        USI00400
   CALL NEWBUF (EXECBUFF,TRNSBUFF,FIRSTLAST);                           USI00410
   IF ^FIRSTLAST THEN RETURN;                                           USI00420
   CALL XBUFF (EXECBUFF);                                               USI00430
END;                                                                    USI00440
                                                                        USI00450
                                                                        USI00460
```

```
XBUFF: PROC (EXECBUFF);                                              USI00470
DCL EXECBUFF CHAR (4000) VAR;                                        USI00480
DCL (UNIT,RETSTMT) CHAR (2000) VAR;                                  USI00490
DCL TKLSPTR PTR;                                                     USI00500
JUNK = DCTNRY (DICTION,'ADHCNTRCLR');                                USI00510
TKLSPTR = NULL ();                                                   USI00520
DO WHILE (INDEX (EXECBUFF, ';') ^= 0);                               USI00530
   TKLSPTR = NULL();                                                 USI00540
   UNIT = GETS (EXECBUFF, ';') || ';' ;                              USI00550
   IF INDEX (UNIT, 'RETRIEVE') ^= 0 THEN DO;                         USI00560
      CALL CMSCMD (PLIST2,RETCODE);                                  USI00570
      RETSTMT = UNIT ;                                               USI00580
      CALL RETDSPLY (RETSTMT) ;                                      USI00590
      END;                                                           USI00600
   CALL ACTCRD (UNIT,TKLSPTR,DICTION);                               USI00610
   END;                                                              USI00620
END XBUFF;                                                           USI00630
                                                                     USI00640
                                                                     USI00650
                                                                     USI00660
                                                                     USI00670
                                                                     USI00680
                                                                     USI00690
```

```
                                                                    USI00700
                                                                    USI00710
                                                                    USI00720
                                                                    USI00730
                                                                    USI00740
GETS: PROC (LIST,TERM_ITEM) RETURNS (CHAR (2000) VAR);              USI00750
DCL LIST CHAR (*) VAR,                                              USI00760
    TERM_ITEM CHAR (1),                                             USI00770
    RTN_LIST CHAR (2000) VAR,                                       USI00780
    I FIXED;                                                        USI00790
                                                                    USI00800
I = INDEX (LIST,TERM_ITEM);                                         USI00810
IF I = 0 THEN DO;                                                   USI00820
   RTN_LIST = LIST;                                                 USI00830
   LIST = '';                                                       USI00840
   END;                                                             USI00850
ELSE DO;                                                            USI00860
   RTN_LIST = SUBSTR (LIST,1,I-1);                                  USI00870
   LIST = SUBSTR (LIST,I+1);                                        USI00880
   END;                                                             USI00890
RETURN (RTN_LIST);                                                  USI00900
END GETS;                                                           USI00910
                                                                    USI00920
                                                                    USI00930
```

```
PRINTT: PROC (HEAD);                                                    USI00940
/* DEBUG TOOL TO PRINT OUT TOKEN CHAIN */                               USI00950
DCL (HEAD,M) PTR;                                                       USI00960
                                                                        USI00970
M = HEAD;                                                               USI00980
DO WHILE (M ^= NULL ());                                                USI00990
   PUT SKIP EDIT ('TEST_TOKEN =$',M -> TOKEN.ITEM,'$$',                 USI01000
                  'TEST_CLASS =$',M -> TOKEN.CLASS,'$$')                USI01010
                 (A,A,A,SKIP,A,A,A);                                    USI01020
   M = M -> TOKEN.NEXT;                                                 USI01030
   END;                                                                 USI01040
                                                                        USI01050
END PRINTT;                                                             USI01060
                                                                        USI01070
                                                                        USI01080
                                                                        USI01090
                                                                        USI01100
                                                                        USI01110
                                                                        USI01120
                                                                        USI01130
```

```
RETDSPLY: PROC (DFNTN);                                                  USIO1140
DCL DFNTN CHAR (2000) VAR;                                               USIO1150
DCL LINE CHAR (2000) VAR;                                                USIO1160
DCL I FIXED;                                                             USIO1170
DCL SPACES FIXED INIT (0);                                               USIO1180
DCL SPACECH CHAR (2) VAR;                                                USIO1190
DO WHILE (DFNTN ^= '');                                                  USIO1200
   LINE = GETS (DFNTN,'@');                                              USIO1210
   IF LINE ^= '' THEN DO;                                                USIO1220
      CALL REPLACE (LINE,'%1',''''''');                                  USIO1230
      CALL REPLACE (LINE,'%2',';');                                      USIO1240
      CALL REPLACE (LINE,'%3',':');                                      USIO1250
      CALL REPLACE (LINE,'%5','@');                                      USIO1260
      CALL REPLACE (LINE,'%4','%');                                      USIO1270
      IF SPACES ^= 0 THEN                                                USIO1280
         DO I = 1 TO SPACES;                                             USIO1290
            LINE = ' ' || LINE;                                          USIO1300
            END;                                                         USIO1310
      PUT EDIT (LINE) (COL(1),A(80));                                    USIO1320
      IF DFNTN ^= '' THEN                                                USIO1330
         DFNTN = '@' || DFNTN;                                           USIO1340
      END;                                                               USIO1350
   ELSE DO;                                                              USIO1360
      SPACECH = GETS (DFNTN,':');                                        USIO1370
      SPACES = FIXED (SPACECH);                                          USIO1380
      END;                                                               USIO1390
   END;                                                                  USIO1400
END RETDSPLY;                                                            USIO1410
                                                                         USIO1420
                                                                         USIO1430
                                                                         USIO1440
                                                                         USIO1450
                                                                         USIO1460
                                                                         USIO1470
```

```
                                                                    USI01480
                                                                    USI01490
                                                                    USI01500
                                                                    USI01510
                                                                    USI01520
REPLACE: PROC (RLINE,FR_SYM,TO_SYM);                               USI01530
DCL (RLINE,REP_LINE) CHAR (2000) VAR,                               USI01540
    FR_SYM CHAR (2) VAR,                                            USI01550
    TO_SYM CHAR (2) VAR,                                            USI01560
    I FIXED;                                                        USI01570
                                                                    USI01580
REP_LINE = '';                                                      USI01590
I = INDEX (RLINE,FR_SYM);                                           USI01600
DO WHILE (I ^= 0);                                                  USI01610
   IF LENGTH (FR_SYM) = 1 THEN DO;                                  USI01620
      REP_LINE= REP_LINE || SUBSTR (RLINE,1,I - 1) || TO_SYM;       USI01630
      RLINE = SUBSTR (RLINE,I + 1);                                 USI01640
      I = INDEX (RLINE,FR_SYM);                                     USI01650
      END;                                                          USI01660
   ELSE DO;                                                         USI01670
      REP_LINE = REP_LINE || SUBSTR (RLINE,1,I-1) || TO_SYM;        USI01680
      RLINE = SUBSTR (RLINE,I+2);                                   USI01690
      I = INDEX (RLINE,FR_SYM);                                     USI01700
      END;                                                          USI01710
   END;                                                             USI01720
      RLINE = REP_LINE || RLINE;                                    USI01730
                                                                    USI01740
END REPLACE;                                                        USI01750
                                                                    USI01760
END USINT;                                                          USI01770
```

```
                                                                        NEW00010
                                                                        NEW00020
                                                                        NEW00030
                                                                        NEW00040
                                                                        NEW00050
NEWBUF: PROC (EXECBUFF,TRNSBUFF,FIRSTLAST);                             NEW00060
                                                                        NEW00070
/*****************************************************************/     NEW00080
/* THIS PROGRAM INTERACTS WITH THE USER WITHIN THE VIRTUAL       */     NEW00090
/* INFORMATION SUBSYSTEM; IT IS CALLED BY THE GENERAL USER       */     NEW00100
/* INTERFACE PROGRAM WHICH ASKS THE USER WHETHER HE WISHES TO    */     NEW00110
/* ENTER THE VIRTUAL INFO SUBSYSTEM OR THE REAL INFO SUBSYSTEM   */     NEW00120
/*****************************************************************/     NEW00130
                                                                        NEW00140
/* FORMAL PARAMETERS EXECBUFF AND TRNSBUFF REPRESENT             */     NEW00150
/* THE EXECUTION AND TRANSACTION BUFFERS. ON A "RUNTRANS" OR     */     NEW00160
/* "DODEL" COMMAND THE TRANSACTION BUFFER WILL BE PUT INTO THE   */     NEW00170
/* EXECUTION BUFFER, AND THE ROUTINE RETURNS. OTHERWISE          */     NEW00180
/* THE ROUTINE WILL RETURN AS SOON AS THERE IS AN EXECUTABLE     */     NEW00190
/* STATEMENT WITHIN THE EXECUTION BUFFER .                       */     NEW00200
/* FORMAL PARAMETER "FIRSTLAST" REPRESENTS THE BIT WHICH         */     NEW00210
/* WILL BE TURNED ON WHEN THE "TERMINATE" COMMAND IS ISSUED      */     NEW00220
/* AND WHICH WILL BE CHECKED BY THE "USER" PROGRAM TO            */     NEW00230
/* TERMINATE SUBSEQUENT CALLS TO THE "BUFFER" PROGRAM .          */     NEW00240
/* BUFFER COMMANDS SUCH AS "FINPUT","FSAVE","TRANSACTION"        */     NEW00250
/* "END", ..... WILL BE HANDLED ENTIRELY WITHIN THIS             */     NEW00260
/* ROUTINE; THUS, THESE COMMANDS WILL NOT BE INCLUDED            */     NEW00270
/* WITHIN THE "BUFFER" PARAMETER WHICH IS PASSED BACK TO THE     */     NEW00280
/* "USINT" PROGRAM .                                             */     NEW00290
/* THIS BUFFER PROGRAM FEATURES AN INTERACTIVE LINE EDITOR       */     NEW00300
/* THROUGH WHICH A USER ACCESSES THE TRANSACTION BUFFER          */     NEW00310
                                                                        NEW00320
DCL PLIST1(3) CHAR(8) INIT ('DVHUTL','TEST',HIGH(8)),                   NEW00330
    PLIST2(3) CHAR(8) INIT ('DVHUTL','CLEAR',HIGH(8)),                  NEW00340
    (HIGH, PLIRETC) BUILTIN,                                            NEW00350
    RETCODE FIXED BINARY (31,0),  /* TO CLEAR SCREEN */                 NEW00360
    CMSCMD EXTERNAL ENTRY OPTIONS (ASSEMBLER INTER);                    NEW00370
DCL (EXECBUFF,TRNSBUFF) CHAR (4000) VAR;                                NEW00380
DCL (FIRSTLAST,STATUS) BIT (1);                                         NEW00390
DCL (WORD,GARBAGE,KEY) CHAR (80) VAR;                                   NEW00400
DCL PRSTLINE CHAR (80);                                                 NEW00410
DCL (CPLNVAR,STRNSP) CHAR (80) VAR;                                     NEW00420
DCL (BLANKLINE,ERROR) BIT(1) INIT ('0'B);                               NEW00430
DCL (I,I1,LDSPACES) FIXED;                                              NEW00440
DCL TRFILE FILE RECORD                                                  NEW00450
    ENV (FB RECSIZE (80) BLKSIZE (800));                                NEW00460
DCL L_OF_CMDS (13) CHAR (12) VAR;                                       NEW00470
DCL LIST_OF_CMDS (13) CHAR (12) VAR                                     NEW00480
    INIT ('RUNTRANS',                                                   NEW00490
          'FSAVE',                                                      NEW00500
          'TERMINATE',                                                  NEW00510
          'TRANSACTION',                                                NEW00520
          'ENDTRANS',                                                   NEW00530
          'FINPUT',                                                     NEW00540
          'KILLEXEC',                                                   NEW00550
          'ERASETRANS',                                                 NEW00560
          'DODELTRANS',                                                 NEW00570
          'HELP',                                                       NEW00580
```

```
          'INSERT',                                                    NEW00590
          'DELETE',                                                    NEW00600
          'TOPLINE');                                                  NEW00610
DCL EXECB BIT (1) INIT ('1'B);                                         NEW00620
DCL (LINENM,TOPLINE) FIXED;                                            NEW00630
DCL 1 EXECMKS,                                                         NEW00640
      2 LIMIT FIXED,                                                   NEW00650
      2 LOC (0:100) FIXED;                                             NEW00660
DCL 1 TRANSMKS LIKE EXECMKS;                                           NEW00670

EXECB = '1'B;                                                          NEW00680
L_OF_CMDS (*) = LIST_OF_CMDS (*);                                      NEW00690
CALL SETMKS (TRNSBUFF,TRANSMKS);                                       NEW00700
CALL SETMKS (EXECBUFF,EXECMKS);                                        NEW00710
TOPLINE = 0;                                                           NEW00720
DO WHILE ('1'B);                                                       NEW00730
ERROR = '0'B;                                                          NEW00740
CALL CMSCMD (PLIST2,RETCODE);          /* CLEAR CREEEN      */         NEW00750
PUT SKIP EDIT                                                          NEW00760
   ('***TRANSACTION BUFFER***')                                        NEW00770
   (COL (25),A);                                                       NEW00780
IF EXECB THEN DO;                                                      NEW00790
   CALL BDISPLAY (TRNSBUFF,TOPLINE,5,TRANSMKS,'1'B);                   NEW00800
   PUT SKIP EDIT                                                       NEW00810
      ('***EXECUTION BUFFER***')                                       NEW00820
      (COL (25),A);                                                    NEW00830
   CALL BDISPLAY (EXECBUFF,0,12,EXECMKS,'0'B);                         NEW00840
   END;                                                                NEW00850
ELSE                                                                   NEW00860
   CALL BDISPLAY (TRNSBUFF,TOPLINE,18,TRANSMKS,'1'B);                  NEW00870
GET EDIT (PRSTLINE) (COL(1),A(80));                                    NEW00880
CPLNVAR = TRANSLATE (PRSTLINE, 'ABCDEFGHIJKLMNOPQRSTUVWXYZ',          NEW00890
                               'abcdefghijklmnopqrstuvwxyz');          NEW00900
CALL REPLACE (CPLNVAR,'%','%4');                                       NEW00910
CALL REPLACE (CPLNVAR,',','%0');                                       NEW00920
CALL REPLACE (CPLNVAR,':','%3');                                       NEW00930
CALL REPLACE (CPLNVAR,'@','%5');                                       NEW00940

LDSPACES = 0;                                                          NEW00950
BLANKLINE = '0'B;                                                      NEW00960
CALL KBLKS;                                                            NEW00970
KEY = NEXTWORD;                                                        NEW00980
IF ERROR THEN GOTO ENDLOOP;                                            NEW00990
IF LENGTH (KEY) >= 1 THEN                                              NEW01000
   L_OF_CMDS (*) = SUBSTR (LIST_OF_CMDS (*),                           NEW01010
                           1,                                          NEW01020
                           MIN (LENGTH (KEY),                          NEW01030
                                LENGTH (LIST_OF_CMDS (*))));           NEW01040

SELECT (KEY);                                                          NEW01050
   WHEN (L_OF_CMDS (1)) DO;                                            NEW01060
      EXECBUFF = TRNSBUFF;                                             NEW01070
      RETURN;                                                          NEW01080
      END;                                                             NEW01090
```

```
WHEN (L_OF_CMDS (2))                                                    NEWO1170
   CALL FSAVE;                                                          NEWO1180
WHEN (L_OF_CMDS (3)) DO;                                                NEWO1190
   FIRSTLAST = '0'B;                                                    NEWO1200
   RETURN;                                                              NEWO1210
   END;                                                                 NEWO1220
WHEN (L_OF_CMDS (4)) DO;                                                NEWO1230
   STRNSP = '';                                                         NEWO1240
   EXECB = '0'B;                                                        NEWO1250
   END;                                                                 NEWO1260
WHEN (L_OF_CMDS (5)) DO;                                                NEWO1270
   EXECB = '1'B;                                                        NEWO1280
   STRNSP = '';                                                         NEWO1290
   END;                                                                 NEWO1300
WHEN (L_OF_CMDS (6))                                                    NEWO1310
   CALL FINPUT;                                                         NEWO1320
WHEN (L_OF_CMDS (7)) DO;                                                NEWO1330
   EXECBUFF = '';                                                       NEWO1340
   CALL SETMKS (EXECBUFF,EXECMKS);                                      NEWO1350
   END;                                                                 NEWO1360
WHEN (L_OF_CMDS (8)) DO;                                                NEWO1370
   TRNSBUFF = '' ;                                                      NEWO1380
   CALL SETMKS (TRNSBUFF,TRANSMKS);                                     NEWO1390
   END;                                                                 NEWO1400
WHEN (L_OF_CMDS (9)) DO;                                                NEWO1410
   EXECBUFF = TRNSBUFF;                                                 NEWO1420
   TRNSBUFF = '' ;                                                      NEWO1430
   RETURN;                                                              NEWO1440
   END;                                                                 NEWO1450
WHEN (L_OF_CMDS (10))                                                   NEWO1460
   CALL HELP;                                                           NEWO1470
WHEN (L_OF_CMDS (11))                                                   NEWO1480
   IF WTHNLM (NEXTWORD,TRANSMKS.LIMIT,LINENM) THEN DO;                  NEWO1490
      CALL KBLKS;                                                       NEWO1500
      STATUS = BUILDBUF (TRNSBUFF,TRANSMKS,LINENM);                     NEWO1510
      END;                                                              NEWO1520
   ELSE                                                                 NEWO1530
   /* MSG */ ;                                                         NEWO1540
WHEN (L_OF_CMDS (12))                                                   NEWO1550
   IF WTHNLM (NEXTWORD,TRANSMKS.LIMIT - 1,LINENM) THEN                  NEWO1560
      CALL DELTE;                                                       NEWO1570
   ELSE /* MSG */ ;                                                    NEWO1580
WHEN (L_OF_CMDS (13))                                                   NEWO1590
   IF WTHNLM (NEXTWORD,TRANSMKS.LIMIT,LINENM) THEN                      NEWO1600
      TOPLINE = LINENM;                                                 NEWO1610
   ELSE /* MSG */ ;                                                    NEWO1620
OTHERWISE DO;                                                           NEWO1630
   CPLNVAR = STRNSP;                                                    NEWO1640
   IF EXECB THEN                                                        NEWO1650
      IF BUILDBUF (EXECBUFF,EXECMKS,EXECMKS.LIMIT) THEN                 NEWO1660
         RETURN;                                                        NEWO1670
                                                                        NEWO1680
                                                                        NEWO1690
                                                                        NEWO1700
                                                                        NEWO1710
                                                                        NEWO1720
```

```
                                                                    NEWO1730
                                                                    NEWO1740
                                                                    NEWO1750
                                                                    NEWO1760
                                                                    NEWO1770
         ELSE;                                                      NEWO1780
      ELSE                                                          NEWO1790
         STATUS = BUILDBUF (TRNSBUFF,TRANSMKS,TRANSMKS.LIMIT);      NEWO1800
      END;                                                          NEWO1810
   END;                                                             NEWO1820
ENDLOOP:                                                            NEWO1830
END;                                                                NEWO1840
```

```
LDSPCH: PROC (LDSPACES) RETURNS (CHAR (20) VAR);                    NEW01850
/* THIS ROUTINE CONSTRUCTS THE LINE HEADER FOR EACH */              NEW01860
/* BUFFER LINE, CHAR "@" || NUMBER OF LEADING SPACES || CHAR ":" */ NEW01870
DCL (LDSPACES,I) FIXED;                                             NEW01880
DCL LDSPCHAR CHAR (20) VAR;                                         NEW01890
LDSPCHAR =  CHAR (LDSPACES) ;                                       NEW01900
DO I = 1 TO LENGTH (LDSPCHAR);                                      NEW01910
   IF SUBSTR (LDSPCHAR,I,1) ^= ' ' THEN DO;                         NEW01920
      LDSPCHAR = SUBSTR (LDSPCHAR,I);                               NEW01930
      I = 100;                                                      NEW01940
      END;                                                          NEW01950
END;                                                                NEW01960
LDSPCHAR = '@' || LDSPCHAR || ':' ;                                 NEW01970
RETURN (LDSPCHAR);                                                  NEW01980
END LDSPCH;                                                         NEW01990
                                                                    NEW02000
                                                                    NEW02010
                                                                    NEW02020
                                                                    NEW02030
                                                                    NEW02040
                                                                    NEW02050
                                                                    NEW02060
```

```
BUILDBUF: PROC (BUFF,MKS,LNUM) RETURNS (BIT (1));                    NEW02070
/* THIS ROUTINE BUILDS UP EITHER THE EXECUTION OR THE */             NEW02080
/* TRANSACTION BUFFER ONE LINE AT A TIME              */             NEW02090
DCL BUFF CHAR (4000) VAR;                                            NEW02100
DCL 1 MKS,                                                           NEW02110
    2 LIMIT FIXED,                                                   NEW02120
    2 LOC (0:100) FIXED;                                             NEW02130
DCL (LNUM,LNUM1,LEN,I) FIXED;                                        NEW02140
CPLNVAR = LDSPCH (LDSPACES) || STRNSP || ' ';                        NEW02150
CALL TRNSLATE;                                                       NEW02160
LEN = LENGTH (CPLNVAR);                                              NEW02170
IF LEN = 0 THEN                                                      NEW02180
   RETURN ('0'B);                                                    NEW02190
LNUM1 = LNUM;                                                        NEW02200
MKS.LIMIT = MKS.LIMIT + 1;                                           NEW02210
DO I = MKS.LIMIT TO LNUM1 + 1 BY -1;                                 NEW02220
   MKS.LOC (I) = MKS.LOC (I-1) + LEN;                                NEW02230
   END;                                                              NEW02240
BUFF = SUBSTR (BUFF,1,MKS.LOC (LNUM1) - 1)                           NEW02250
          || CPLNVAR                                                 NEW02260
          || SUBSTR (BUFF,MKS.LOC (LNUM1));                          NEW02270
RETURN (INDEX (CPLNVAR,';') ^= 0);                                   NEW02280
END BUILDBUF;                                                        NEW02290
                                                                     NEW02300
                                                                     NEW02310
                                                                     NEW02320
                                                                     NEW02330
                                                                     NEW02340
                                                                     NEW02350
```

```
                                                                    NEW02360
                                                                    NEW02370
                                                                    NEW02380
                                                                    NEW02390
                                                                    NEW02400
TRNSLATE: PROC;  /* STATUS BEING 'O'B UPON ENTRY                   */ NEW02410
/* THIS PROGRAM CHANGES QUOTE AND SEMICOLON CHARACTERS WITHIN      */ NEW02420
/* STRING CONSTANTS TO SPECIAL CHARACTERS, AND ALSO THE SEMICOLON */ NEW02430
/* CHARACTERS WITHIN "COMMENT STATEMENTS TO A SPECIAL SYMBOL      */ NEW02440
/* ROUTINE ACTS ON GLOBAL VARIABLE "CPLNVAR"                      */ NEW02450
DCL (TEMP1,COPY) CHAR (80) VAR INIT ('');                           NEW02460
DCL (QUOTE, QUOTE1, SLSH, SLSH1, SCOLON, J) FIXED (4);              NEW02470
DO WHILE ('1'B);                                                    NEW02480
   SLSH = INDEX (CPLNVAR, '\');                                     NEW02490
   QUOTE = INDEX (CPLNVAR, '''');                                   NEW02500
   SCOLON = INDEX (CPLNVAR, ';');                                   NEW02510
   IF SLSH=0 & QUOTE=0 THEN GOTO DONE;                              NEW02520
   IF QUOTE^=0 & (SLSH=0 ¦ QUOTE<SLSH) THEN                         NEW02530
      DO;                                                           NEW02540
         COPY = COPY ¦¦ SUBSTR (CPLNVAR,1,QUOTE);                   NEW02550
         TEMP1 = SUBSTR (CPLNVAR,QUOTE+1);                          NEW02560
         DO WHILE (INDEX (TEMP1, '''''') ^= 0);                     NEW02570
            IF INDEX (TEMP1, '''''') = 1                            NEW02580
               THEN TEMP1 = '%1' ¦¦ SUBSTR (TEMP1,3);               NEW02590
               ELSE TEMP1 = SUBSTR (TEMP1,1,INDEX (TEMP1,'''''') - 1) NEW02600
                            ¦¦ '%1' ¦¦                              NEW02610
                            SUBSTR (TEMP1, INDEX (TEMP1,'''''') + 2); NEW02620
         END;                                                       NEW02630
         QUOTE1 = INDEX (TEMP1,'''');                               NEW02640
         IF QUOTE1 = 0 THEN                                         NEW02650
            DO;                                                     NEW02660
               CPLNVAR = '';                                        NEW02670
               COPY = '';                                           NEW02680
               CALL CMSCMD (PLIST2,RETCODE);                        NEW02690
               PUT SKIP EDIT ('MISSING QUOTE TERMINATOR') (A);      NEW02700
               PUT SKIP EDIT ('TYPE "ENTER" KEY TO CONTINUE')(A);   NEW02710
               GET EDIT (GARBAGE) (A(80));                          NEW02720
               GOTO DONE;                                           NEW02730
            END;                                                    NEW02740
         TEMP1 = SUBSTR (TEMP1,1,QUOTE1);                          NEW02750
         DO J = 1 TO QUOTE1;                                        NEW02760
            IF SUBSTR (TEMP1,J,1) = ';'                             NEW02770
               THEN COPY = COPY ¦¦ '%2' ;                           NEW02780
               ELSE COPY = COPY ¦¦ SUBSTR (TEMP1,J,1);              NEW02790
         END;                                                       NEW02800
         CPLNVAR = SUBSTR (CPLNVAR,QUOTE+QUOTE1+1);                 NEW02810
      END;                                                          NEW02820
                                                                    NEW02830
   IF SLSH^=0 & (QUOTE=0 ¦ SLSH<QUOTE) THEN                         NEW02840
      DO;                                                           NEW02850
         COPY = COPY ¦¦ SUBSTR (CPLNVAR,1,SLSH);                    NEW02860
         TEMP1 = SUBSTR (CPLNVAR,SLSH+1);                          NEW02870
         SLSH1 = INDEX (TEMP1,'\');                                 NEW02880
         IF SLSH1 = 0 THEN DO;                                      NEW02890
            COPY = '';                                              NEW02900
            CPLNVAR = '';                                           NEW02910
            CALL CMSCMD (PLIST2,RETCODE);                           NEW02920
            PUT SKIP EDIT ('*ERROR* MISSING COMMENT TERMINATOR') (A); NEW02930
            PUT SKIP EDIT ('TYPE "ENTER" KEY TO CONTINUE') (A);     NEW02940
```

```
                                                                    NEW02950
                                                                    NEW02960
                                                                    NEW02970
                                                                    NEW02980
                                                                    NEW02990
            GET EDIT (GARBAGE) (A(80));                             NEW03000
            RETURN;                                                 NEW03010
            END;                                                    NEW03020
         TEMP1 = SUBSTR (TEMP1,1,SLSH1);                            NEW03030
         DO J = 1 TO SLSH1;                                         NEW03040
            IF SUBSTR (TEMP1,J,1) = ';'                             NEW03050
               THEN COPY = COPY || '%2' ;                           NEW03060
               ELSE COPY = COPY || SUBSTR (TEMP1,J,1);              NEW03070
         END;                                                       NEW03080
         CPLNVAR = SUBSTR (CPLNVAR, SLSH+SLSH1+1);                  NEW03090
      END;                                                          NEW03100
END;                                                                NEW03110
                                                                    NEW03120
                                                                    NEW03130
DONE:                                                               NEW03140
   CPLNVAR = COPY || CPLNVAR ;                                      NEW03150
END TRNSLATE;                                                       NEW03160
                                                                    NEW03170
                                                                    NEW03180
                                                                    NEW03190
```

```
FINPUT: PROC;                                                          NEW03200
DCL PRSTLINE CHAR (80);                                                NEW03210
DCL TRFILE FILE RECORD                                                 NEW03220
    ENV (FB RECSIZE (80) BLKSIZE (800));                               NEW03230
DCL FNAME CHAR (80) VAR;                                               NEW03240
ON ENDFILE (TRFILE) GOTO FINEND;                                       NEW03250
ON TRANSMIT (TRFILE) GOTO ERRPT;                                       NEW03260
CALL CMSCMD (PLIST2,RETCODE);                                          NEW03270
FNAME = NEXTWORD;                                                      NEW03280
FNAME = TRANSLATE (FNAME,                                              NEW03290
        '                                  ',                          NEW03300
        '~¦¢#%$^&*()_+=-¢\¦!{}''";:/?.,<>');                           NEW03310
IF INDEX (FNAME,'.') ^= 0                                              NEW03320
   ¦ ERROR                                                             NEW03330
   ¦ LENGTH (FNAME) > 8 THEN                                           NEW03340
ERRPT: DO;                                                             NEW03350
   PUT SKIP EDIT                                                       NEW03360
      ('INVALID FILE TYPE! FILE NOT ENTERED INTO BUFFER') (A);         NEW03370
   PUT SKIP EDIT                                                       NEW03380
      ('TYPE "ENTER" KEY TO CONTINUE') (A);                            NEW03390
   GET EDIT (GARBAGE) (COL(1),A(80));                                  NEW03400
   RETURN;                                                             NEW03410
   END;                                                                NEW03420
PUT SKIP EDIT ('READING TRANSACTION BUFFER FROM CMS FILE:') (A);       NEW03430
PUT SKIP EDIT ('"FILE ' ¦¦ FNAME ¦¦ '"') (COL (30),A);                 NEW03440
PUT SKIP EDIT ('OLD TRANSACTION BUFFER CONTENT DELETED') (A);          NEW03450
CALL WAIT;                                                             NEW03460
OPEN FILE (TRFILE) TITLE (FNAME) INPUT;                                NEW03470
TRNSBUFF = '';                                                         NEW03480
DO WHILE ('1'B);                                                       NEW03490
   READ FILE (TRFILE) INTO (PRSTLINE);                                 NEW03500
   CPLNVAR = TRANSLATE (PRSTLINE, 'ABCDEFGHIJKLMNOPQRSTUVWXYZ',        NEW03510
                                  'ABCDEFGHIJKLMNOPQRSTUVWXYZ');       NEW03520
   CALL REPLACE (CPLNVAR,'%','%4');                                    NEW03530
   CALL REPLACE (CPLNVAR,'.','%O');                                    NEW03540
   CALL REPLACE (CPLNVAR,':','%3');                                    NEW03550
   CALL REPLACE (CPLNVAR,'¢','%5');                                    NEW03560
   LDSPACES = 0;                                                       NEW03570
   BLANKLINE = '0'B;                                                   NEW03580
   CALL KBLKS;                                                         NEW03590
   CPLNVAR =  LDSPCH (LDSPACES) ¦¦ STRNSP ;                            NEW03600
   CALL TRNSLATE;                                                      NEW03610
   TRNSBUFF = TRNSBUFF ¦¦ CPLNVAR ;                                    NEW03620
   END;                                                                NEW03630
FINEND:                                                                NEW03640
CALL SETMKS (TRNSBUFF,TRANSMKS);                                       NEW03650
CLOSE FILE (TRFILE);                                                   NEW03660
END FINPUT;                                                            NEW03670
                                                                       NEW03680
                                                                       NEW03690
                                                                       NEW03700
                                                                       NEW03710
                                                                       NEW03720
                                                                       NEW03730
```

```
                                                                        NEW03740
                                                                        NEW03750
                                                                        NEW03760
                                                                        NEW03770
                                                                        NEW03780
KBLKS: PROC;                                                            NEW03790
/* SETS UP "STRNSP" FROM "CPLNVAR" */                                   NEW03800
/* CALCULATES "LDSPACES"           */                                   NEW03810
DCL I FIXED;                                                            NEW03820
DO I = 1 TO LENGTH (CPLNVAR);                                           NEW03830
   IF SUBSTR (CPLNVAR,I,1) ^= ' '                                       NEW03840
      THEN DO;                                                          NEW03850
              LDSPACES = I - 1;                                         NEW03860
              I = 150;                                                  NEW03870
           END;                                                         NEW03880
      ELSE                                                              NEW03890
           IF I = LENGTH (CPLNVAR) THEN BLANKLINE = '1'B;               NEW03900
END;                                                                    NEW03910
IF BLANKLINE = '0'B                                                     NEW03920
   THEN DO;                                                             NEW03930
           CPLNVAR = SUBSTR (CPLNVAR, LDSPACES + 1);                    NEW03940
           DO I = LENGTH (CPLNVAR) TO 1 BY -1;                          NEW03950
              IF SUBSTR (CPLNVAR,I,1) ^= ' '                            NEW03960
                 THEN DO;                                               NEW03970
                         CPLNVAR = SUBSTR (CPLNVAR,1,I);                NEW03980
                         I = 0;                                         NEW03990
                      END;                                              NEW04000
           END;                                                         NEW04010
           STRNSP = CPLNVAR;                                            NEW04020
        END;                                                            NEW04030
   ELSE DO;                                                             NEW04040
           STRNSP = '';                                                 NEW04050
           LDSPACES = 0;                                                NEW04060
        END;                                                            NEW04070
END KBLKS;                                                              NEW04080
```

```
GETS: PROC (LIST,TERM_ITEM) RETURNS (CHAR (80) VAR);                NEW04090
DCL LIST CHAR (*) VAR,                                              NEW04100
    TERM_ITEM CHAR (1),                                             NEW04110
    RTN_LIST CHAR (80) VAR,                                         NEW04120
    I FIXED;                                                        NEW04130
                                                                    NEW04140
I = INDEX (LIST,TERM_ITEM);                                         NEW04150
IF I = 0 THEN DO;                                                   NEW04160
   RTN_LIST = LIST;                                                 NEW04170
   LIST = '';                                                       NEW04180
   END;                                                             NEW04190
ELSE DO;                                                            NEW04200
   RTN_LIST = SUBSTR (LIST,1,I - 1);                                NEW04210
   LIST = SUBSTR (LIST,I + 1);                                      NEW04220
   END;                                                             NEW04230
RETURN (RTN_LIST);                                                  NEW04240
END GETS;                                                           NEW04250
                                                                    NEW04260
                                                                    NEW04270
                                                                    NEW04280
                                                                    NEW04290
                                                                    NEW04300
```

```
                                                                    NEW04310
                                                                    NEW04320
                                                                    NEW04330
                                                                    NEW04340
                                                                    NEW04350
NEXTWORD: PROC RETURNS (CHAR (80) VAR);                             NEW04360
/* THIS ROUTINE RETURNS A NULL STRING IF CPLNVAR IS NULL,      */   NEW04370
/* RETURNS THE STRING '' IF CPLNVAR IS A BLANKLINE,            */   NEW04380
/* OTHERWISE, IT RETURNS THE FIRST WORD OF CPLNVAR             */   NEW04390
/* DELIMITED BY AN OPTIONAL SUCCEEDING BLANKLINE CHARACTER     */   NEW04400
/*                                                             */   NEW04410
DCL NWORD CHAR (80) VAR;                                            NEW04420
DCL SYMBOL CHAR (1);                                                NEW04430
IF CPLNVAR = '' THEN RETURN ('');                                   NEW04440
DO WHILE ('1'B);                                                    NEW04450
   CALL RMVFBLKS;                                                   NEW04460
   IF CPLNVAR = '' THEN RETURN ('');                                NEW04470
   IF INDEX ('\@',SUBSTR (CPLNVAR,1,1)) ^= 0 THEN                   NEW04480
      DO WHILE (INDEX ('\@',SUBSTR (CPLNVAR,1,1)) ^= 0);            NEW04490
         SYMBOL = SUBSTR (CPLNVAR,1,1);                             NEW04500
         CPLNVAR = SUBSTR (CPLNVAR,2);                              NEW04510
         IF SYMBOL = '\' THEN                                       NEW04520
            IF INDEX (CPLNVAR,'\') = 0 THEN DO;                     NEW04530
               ERROR = '1';                                         NEW04540
               CPLNVAR = '';                                        NEW04550
               PUT SKIP EDIT ('MISSING COMMENT TERMINATOR') (A);    NEW04560
               PUT SKIP EDIT ('TYPE "ENTER" TO CONTINUE') (A);      NEW04570
               GET EDIT (GARBAGE) (A(80));                          NEW04580
               RETURN;                                              NEW04590
               END;                                                 NEW04600
            ELSE                                                    NEW04610
               GARBAGE = GETS (CPLNVAR,SYMBOL);                     NEW04620
         ELSE                                                       NEW04630
            GARBAGE = GETS (CPLNVAR,':');                           NEW04640
         END;                                                       NEW04650
   IF SUBSTR (CPLNVAR,1,1) ^= ' ' THEN DO;                          NEW04660
      NWORD = GETS (CPLNVAR, ' ');                                  NEW04670
      RETURN (NWORD);                                               NEW04680
      END;                                                          NEW04690
END;                                                                NEW04700
```

```
RMVFBLKS: PROC;                    /* REMOVE FRONT BLANKS */
DCL I FIXED;
IF CPLNVAR = '' THEN RETURN;
DO I = 1 TO LENGTH (CPLNVAR);
   IF SUBSTR (CPLNVAR,I,1) ^= ' ' THEN DO;
      CPLNVAR = SUBSTR (CPLNVAR, I);
      RETURN;
      END;
   END;
CPLNVAR = '' ;
RETURN;
END RMVFBLKS;
END NEXTWORD;
```

```
NEW04710
NEW04720
NEW04730
NEW04740
NEW04750
NEW04760
NEW04770
NEW04780
NEW04790
NEW04800
NEW04810
NEW04820
NEW04830
NEW04840
NEW04850
NEW04860
NEW04870
NEW04880
```

```
FSAVE: PROC;                                                          NEWO4890
DCL FNAME CHAR (80) VAR;                                              NEWO4900
DCL TRFILE FILE RECORD                                                NEWO4910
    ENV (FB RECSIZE (80) BLKSIZE (800));                              NEWO4920
DCL LDSPCHAR CHAR (20) VAR;                                           NEWO4930
TRNSBUFF = TRNSBUFF || LDSPCH (0);                                    NEWO4940
ON ENDFILE (TRFILE) GOTO FSVEND;                                      NEWO4950
CALL CMSCMD (PLIST2,RETCODE);                                         NEWO4960
FNAME = NEXTWORD;                                                     NEWO4970
FNAME = TRANSLATE (FNAME,                                             NEWO4980
        '.................................',                           NEWO4990
        '~¦@#%$^&*()_+=-¢\¦!{}''":;/?.,<>');                          NEWO5000
IF INDEX (FNAME,'.') ^= 0                                             NEWO5010
   ¦ ERROR                                                            NEWO5020
   ¦ LENGTH (FNAME) > 8 THEN DO;                                      NEWO5030
   PUT SKIP EDIT                                                      NEWO5040
      ('INVALID FILE NAME! BUFFER NOT SAVED') (A);                    NEWO5050
   PUT SKIP EDIT                                                      NEWO5060
      ('TYPE "ENTER" KEY TO CONTINUE') (A);                           NEWO5070
   GET EDIT (GARBAGE) (COL(1),A(80));                                 NEWO5080
   GOTO FSVEND;                                                       NEWO5090
   END;                                                               NEWO5100
PUT SKIP EDIT ('SAVING TRANSACTION BUFFER INTO CMS FILE:') (A);       NEWO5110
PUT SKIP EDIT ('"FILE ' || FNAME || '"') (COL(30),A);                 NEWO5120
CALL WAIT;                                                            NEWO5130
OPEN FILE (TRFILE) TITLE (FNAME) OUTPUT;                              NEWO5140
CPLNVAR = '';                                                         NEWO5150
DO WHILE (TRNSBUFF ^= '');                                            NEWO5160
   TRNSBUFF = SUBSTR (TRNSBUFF,2);                                    NEWO5170
   LDSPCHAR = GETS (TRNSBUFF,':');                                    NEWO5180
   IF SUBSTR (TRNSBUFF,1,1) = '@' THEN                                NEWO5190
      IF LDSPCHAR = '0' THEN DO;                                      NEWO5200
         CPLNVAR = '';                                                NEWO5210
         PRSTLINE = CPLNVAR;                                          NEWO5220
         CPLNVAR = '';                                                NEWO5230
         WRITE FILE (TRFILE) FROM (PRSTLINE);                         NEWO5240
         END;                                                         NEWO5250
      ELSE DO;                                                        NEWO5260
         DO I = 1 TO FIXED (LDSPCHAR);                                NEWO5270
            CPLNVAR = CPLNVAR || ' ' ;                                NEWO5280
            END;                                                      NEWO5290
         PRSTLINE = CPLNVAR;                                          NEWO5300
         CPLNVAR = '';                                                NEWO5310
         WRITE FILE (TRFILE) FROM (PRSTLINE);                         NEWO5320
         END;                                                         NEWO5330
   ELSE IF TRNSBUFF ^= '' THEN DO;                                    NEWO5340
      IF LDSPCHAR ^= '0' THEN                                         NEWO5350
         DO I = 1 TO FIXED (LDSPCHAR);                                NEWO5360
            CPLNVAR = CPLNVAR || ' ' ;                                NEWO5370
            END;                                                      NEWO5380
      CPLNVAR = CPLNVAR || GETS (TRNSBUFF,'@');                       NEWO5390
      TRNSBUFF = '@' || TRNSBUFF;                                     NEWO5400
      CALL REPLACE (CPLNVAR,'%0',',');                                NEWO5410
      CALL REPLACE (CPLNVAR,'%2',';');                                NEWO5420
```

```
      CALL REPLACE (CPLNVAR,'%1',''''); NEW05480
      CALL REPLACE (CPLNVAR,'%3',':'); NEW05490
      CALL REPLACE (CPLNVAR,'%5','*'); NEW05500
      CALL REPLACE (CPLNVAR,'%4','%'); NEW05510
      PRSTLINE = CPLNVAR; NEW05520
      CPLNVAR = ''; NEW05530
      WRITE FILE (TRFILE) FROM (PRSTLINE); NEW05540
      END; NEW05550
END; NEW05560
FSVEND: NEW05570
CLOSE FILE (TRFILE); NEW05580
CALL SETMKS (TRNSBUFF,TRANSMKS); NEW05590
END FSAVE; NEW05600
NEW05610
NEW05620
NEW05630
NEW05640
NEW05650
NEW05660
NEW05670
```

```
WAIT: PROC;                                                    NEW05680
/* THIS PROGRAM SIMPLY DELAYS EXECUTION BY A FEW SECONDS */    NEW05690
/* USED TO HOLD DISPLAY LONG ENOUGH TO BE RECOGNIZED     */    NEW05700
DCL (I,I1) FIXED;                                              NEW05710
DO I = 1 TO 500;                                               NEW05720
   DO I1 = 1 TO 500;                                           NEW05730
   END;                                                        NEW05740
END;                                                           NEW05750
END WAIT;                                                      NEW05760
                                                               NEW05770
                                                               NEW05780
                                                               NEW05790
                                                               NEW05800
                                                               NEW05810
                                                               NEW05820
```

```
                                                                    NEW05830
                                                                    NEW05840
                                                                    NEW05850
                                                                    NEW05860
                                                                    NEW05870
REPLACE: PROC (RLINE,FR_SYM,TO_SYM);                               NEW05880
                                                                    NEW05890
DCL (RLINE,REP_LINE) CHAR (80) VAR,                                 NEW05900
    FR_SYM CHAR (2) VAR,                                            NEW05910
    TO_SYM CHAR (2) VAR,                                            NEW05920
    I FIXED;                                                        NEW05930
                                                                    NEW05940
REP_LINE = '';                                                      NEW05950
I = INDEX (RLINE,FR_SYM);                                           NEW05960
DO WHILE (I ^= 0);                                                  NEW05970
   IF LENGTH (FR_SYM) = 1 THEN DO;                                  NEW05980
      REP_LINE= REP_LINE || SUBSTR (RLINE,1,I - 1) || TO_SYM;       NEW05990
      RLINE = SUBSTR (RLINE,I + 1);                                 NEW06000
      I = INDEX (RLINE,FR_SYM);                                     NEW06010
      END;                                                          NEW06020
   ELSE DO;                                                         NEW06030
      REP_LINE = REP_LINE || SUBSTR (RLINE,1,I-1) || TO_SYM;        NEW06040
      RLINE = SUBSTR (RLINE,I+2);                                   NEW06050
      I = INDEX (RLINE,FR_SYM);                                     NEW06060
      END;                                                          NEW06070
   END;                                                             NEW06080
      RLINE = REP_LINE || RLINE;                                    NEW06090
                                                                    NEW06100
END REPLACE;                                                        NEW06110
```

```
                                                                        NEW06120
                                                                        NEW06130
                                                                        NEW06140
                                                                        NEW06150
                                                                        NEW06160
HELP: PROC;                                                             NEW06170
CALL CMSCMD (PLIST2,RETCODE);                                           NEW06180
PUT SKIP EDIT                                                           NEW06190
   ('"RUNTRANS"      EXECUTE TRANSACTION BUFFER',                       NEW06200
    '"DODELTRANS"    EXECUTE TRANSACTION BUFFER, BUFFER DELETED',       NEW06210
    '"FINPUT"(ARG)   INPUT TRANSACTION BUFFER FROM FILE "FILE (ARG)"',  NEW06220
    '"FSAVE"(ARG)    SAVE TRANSACTION BUFFER INTO FILE "FILE (ARG)"',   NEW06230
    '"TERMINATE"     TERMINATE PROGRAM - RETURN TO CMS',                NEW06240
    '"TRANSACTION"   ENTER CURRENT TRANSACTION BUFFER',                 NEW06250
    '"ENDTRANS"      EXIT TRANSACTION BUFFER',                          NEW06260
    '"ERASETRANS"    ERASE CURRENT TRANSACTION BUFFER',                 NEW06270
    '"KILLEXEC"      KILL CURRENT EXECUTION BUFFER',                    NEW06280
    '"HELP"          BRIEF EXPLANATION OF COMMANDS',                    NEW06290
    '"INSERT"        INSERT "STR" STRING INTO TRANSACTION BUFFER AT',    NEW06300
    '    (NUM)(STR)  LINE NUMBER "NUM"',                                NEW06310
    '"DELETE"(NUM)   DELETE (NUM)TH LINE FROM TRANSACTION BUFFER',      NEW06320
    '"TOPLINE"(NUM)  START DISPLAY FROM (NUM)TH LINE OF BUFFER')        NEW06330
   (14 (SKIP,A));                                                       NEW06340
PUT SKIP EDIT                                                           NEW06350
   ('EACH COMMAND MAY BE IDENTIFIED BY TWO OR MORE OF ITS LEADING' ||   NEW06360
    ' CHARACTERS.')                                                     NEW06370
   (A);                                                                 NEW06380
PUT SKIP EDIT ('TYPE "ENTER" KEY TO CONTINUE') (A);                     NEW06390
GET EDIT (GARBAGE) (A(80));                                             NEW06400
END HELP;                                                               NEW06410
```

```
SETMKS: PROC (BUFF,MKS);                                                NEW06420
DCL BUFF CHAR (4000) VAR;                                               NEW06430
DCL (I,J,LASTMARK,NOWMARK) FIXED;                                       NEW06440
DCL 1 MKS,                                                              NEW06450
      2 LIMIT FIXED,                                                    NEW06460
      2 LOC (0:100) FIXED;                                              NEW06470
MKS.LOC (*) = 0;                                                        NEW06480
J = 0;                                                                  NEW06490
LASTMARK = 0;                                                           NEW06500
NOWMARK = INDEX (BUFF,'@');                                             NEW06510
DO WHILE (NOWMARK ^= 0);                                                NEW06520
   LASTMARK = LASTMARK + NOWMARK;                                       NEW06530
   MKS.LOC (J) = LASTMARK;                                              NEW06540
   J = J + 1;                                                           NEW06550
   NOWMARK = INDEX (SUBSTR (BUFF, LASTMARK + 1),'@');                   NEW06560
   END;                                                                 NEW06570
MKS.LOC (J) = LENGTH (BUFF) + 1;                                        NEW06580
MKS.LIMIT = J;                                                          NEW06590
END SETMKS;                                                             NEW06600
                                                                        NEW06610
                                                                        NEW06620
                                                                        NEW06630
                                                                        NEW06640
                                                                        NEW06650
```

```
BDISPLAY: PROC (BUFF,STARTLN,NUML,MKS,PRTLNM);                          NEW06660
/* THIS PROCEDURE DISPLAYS BUFFER CONTENT */                            NEW06670
DCL BUFF CHAR (4000) VAR;                                               NEW06680
DCL (STARTLN,NUML,LNUM,I,SPC,LNLOC) FIXED;                              NEW06690
DCL 1 MKS,                                                              NEW06700
      2 LIMIT FIXED,                                                    NEW06710
      2 LOC (0:100) FIXED;                                              NEW06720
DCL PRTLNM BIT (1);                                                     NEW06730
DCL PRTLINE CHAR (100) VAR;                                             NEW06740
LNUM = STARTLN;                                                         NEW06750
DO I = 1 TO NUML;                                                       NEW06760
   IF LNUM < MKS.LIMIT THEN DO;                                         NEW06770
      LNLOC = MKS.LOC (LNUM) + 1;                                       NEW06780
      PRTLINE = SUBSTR (BUFF,LNLOC,MKS.LOC (LNUM + 1) - LNLOC);         NEW06790
      SPC = FIXED (GETS (PRTLINE,':'));                                 NEW06800
      IF PRTLNM THEN                                                    NEW06810
         PUT SKIP EDIT (LNUM,PRTLINE)(F(2),X (SPC + 1),A);              NEW06820
      ELSE                                                              NEW06830
         PUT SKIP EDIT (PRTLINE)(X (SPC),A);                            NEW06840
      END;                                                              NEW06850
   ELSE IF LNUM = MKS.LIMIT THEN DO;                                    NEW06860
      IF PRTLNM THEN                                                    NEW06870
         PUT SKIP EDIT (LNUM,'***NO MORE***')(F(2),X (23),A);           NEW06880
      ELSE                                                              NEW06890
         PUT SKIP EDIT ('***NO MORE***')(X(25),A);                      NEW06900
      END;                                                              NEW06910
   ELSE                                                                 NEW06920
      PUT SKIP EDIT ('**')(A);                                          NEW06930
   LNUM = LNUM + 1;                                                     NEW06940
   END;                                                                 NEW06950
END BDISPLAY;                                                           NEW06960
                                                                        NEW06970
                                                                        NEW06980
                                                                        NEW06990
                                                                        NEW07000
                                                                        NEW07010
                                                                        NEW07020
```

```
                                                                    NEW07030
                                                                    NEW07040
                                                                    NEW07050
                                                                    NEW07060
                                                                    NEW07070
DELTE: PROC;                                                        NEW07080
/* THIS IS THE LINE EDITOR FUNCTION DELETE */                       NEW07090
DCL (I,LEN) FIXED;                                                  NEW07100
TRNSBUFF = SUBSTR (TRNSBUFF,1,TRANSMKS.LOC (LINENM)-1)              NEW07110
         || SUBSTR (TRNSBUFF,TRANSMKS.LOC (LINENM + 1));            NEW07120
LEN = TRANSMKS.LOC (LINENM + 1) - TRANSMKS.LOC (LINENM);            NEW07130
TRANSMKS.LIMIT = TRANSMKS.LIMIT - 1;                                NEW07140
DO I = LINENM TO TRANSMKS.LIMIT;                                    NEW07150
   TRANSMKS.LOC (I) = TRANSMKS.LOC (I+1) - LEN;                     NEW07160
   END;                                                             NEW07170
END DELTE;                                                          NEW07180
```

```
                                                                    NEW07190
                                                                    NEW07200
                                                                    NEW07210
                                                                    NEW07220
                                                                    NEW07230
WTHNLM: PROC (ARGSTR,ARGLM,ARGNUM) RETURNS (BIT(1));               NEW07240
/* THIS FUNCTION IS USED TO CHECK VALIDITY OF EDITOR COMMANDS */    NEW07250
DCL ARGSTR CHAR (80) VAR;                                           NEW07260
DCL (ARGLM,ARGNUM) FIXED;                                           NEW07270
ON CONVERSION GOTO BAD;                                             NEW07280
ARGNUM = FIXED (ARGSTR);                                            NEW07290
IF ARGNUM >= 0 & ARGNUM <= ARGLM THEN                               NEW07300
   RETURN ('1'B);                                                   NEW07310
BAD:                                                                NEW07320
RETURN ('0'B);                                                      NEW07330
END WTHNLM;                                                         NEW07340
                                                                    NEW07350
FINISHED:                                                           NEW07360
END NEWBUF;                                                         NEW07370
```

```
                                                                        ACT00010
                                                                        ACT00020
                                                                        ACT00030
                                                                        ACT00040
                                                                        ACT00050
ACTCRD: PROC (UNIT,TKLSPTR,DICTION);                                    ACT00060
                                                                        ACT00070
/*********************************************************************/ACT00080
/* THIS PROGRAM COORDINATES THE ACTIVITIES IN THE VIRTUAL INFORMATION*/ACT00090
/* LEVEL, WHICH IS BELOW THE USER INTERFACE LEVEL.                   */ACT00100
/* IT CALLS THE TOKENIZER, THE PARSER, THE SIMPLIFIER, THE           */ACT00110
/*********************************************************************/ACT00120
                                                                        ACT00130
%INCLUDE DICTION;                                                       ACT00140
%INCLUDE XCHNGE;                                                        ACT00150
%INCLUDE ENTITY;                                                        ACT00160
%INCLUDE TOKEN;                                                         ACT00170
%INCLUDE MACH;                                                          ACT00180
%INCLUDE XTREE;                                                         ACT00190
%INCLUDE RETEARG;                                                       ACT00200
                                                                        ACT00210
DCL UNIT CHAR (2000) VAR;                                               ACT00220
DCL (TKLSPTR,RETEP) PTR;                                                ACT00230
DCL TKNIZE ENTRY EXTERNAL;                                              ACT00240
DCL GO BIT (1) INIT ('1'B);                                             ACT00250
DCL DEFMCH ENTRY EXTERNAL;                                              ACT00260
DCL PARSE ENTRY EXTERNAL;                                               ACT00270
DCL SMPLFY ENTRY EXTERNAL;                                              ACT00280
DCL GARBAGE CHAR (80);                                                  ACT00290
                                                                        ACT00300
CALL TKNIZE (UNIT,TKLSPTR,DICTION,GO);                                  ACT00310
IF ^GO THEN RETURN;                                                     ACT00320
CALL PRINTT;                                                            ACT00330
CALL DEFMCH (MACH);                                                     ACT00340
CALL PARSE (MACH,TKLSPTR,XTREE,XCHNGE,ENTITY,'1'B);                     ACT00350
/* LAST BIT PARAMETER CONTROLS THE DEBUG "TRACE" FEATURE */             ACT00360
/* WHICH SHOWS THE STATE TRANSITIONS AND STACK CONTENTS  */             ACT00370
/* OF THE PUSH-DOWN AUTOMATA, FINITE STATE MACHIN        */             ACT00380
CALL PRINTX;                                                            ACT00390
/* CALL PRINTE; */                                                      ACT00400
CALL SMPLFY (XTREE,ENTITY,XCHNGE,RETEP);                                ACT00410
CALL PRINTR;                                                            ACT00420
PUT EDIT ('TYPE "ENTER" KEY TO CONTINUE') (A);                          ACT00430
GET EDIT (GARBAGE) (A(80));                                             ACT00440
                                                                        ACT00450
```

```
                                                                    ACT00460
                                                                    ACT00470
                                                                    ACT00480
                                                                    ACT00490
                                                                    ACT00500
PRINTT: PROC;    /* DEBUGGING TOOL TO PRINT TOKEN CHAIN */          ACT00510
                                                                    ACT00520
DCL M PTR;                                                          ACT00530
                                                                    ACT00540
M = TKLSPTR;                                                        ACT00550
DO WHILE (M ^= NULL());                                             ACT00560
   PUT SKIP EDIT ('TEST_TOKEN =$',M -> TOKEN.ITEM,'$$',             ACT00570
                 'TEST_CLASS =$',M -> TOKEN.CLASS,'$$')             ACT00580
                (A,A,A,SKIP,A,A,A);                                 ACT00590
   M = M -> TOKEN.NEXT;                                             ACT00600
   END;                                                             ACT00610
                                                                    ACT00620
END PRINTT;                                                         ACT00630
                                                                    ACT00640
```

```
                                                                    ACT00650
                                                                    ACT00660
                                                                    ACT00670
                                                                    ACT00680
                                                                    ACT00690
PRINTM: PROC;  /* DEBUG TOOL TO TRACE THROUGH TRANSITIONS IN */     ACT00700
               /* THE FINITE STATE MACHIN                    */     ACT00710
DCL I FIXED;                                                        ACT00720
                                                                    ACT00730
DO I = 1 TO MACH.STATE_MAP (7) - 1;                                 ACT00740
   PUT SKIP EDIT ('STATE_MAP = ',I,                                 ACT00750
                  'MATCH_STR = ',MACH.MATCH (I),                    ACT00760
                  'ACTIN_STR = ',MACH.ACTION (I),                   ACT00770
                  'NXT_STATE = ',MACH.NEXT_STATE (I))               ACT00780
                 (A,F(5),2 (SKIP,A,A),SKIP,A,F(5));                 ACT00790
   END;                                                             ACT00800
                                                                    ACT00810
DO I = 1 TO 7;                                                      ACT00820
   PUT SKIP EDIT ('STATE = ',I,'   MAP = ',MACH.STATE_MAP (I))      ACT00830
                 (2 (A,F(5)));                                      ACT00840
   END;                                                             ACT00850
END PRINTM;                                                         ACT00860
                                                                    ACT00870
```

```
PRINTX: PROC;  /* DEBUG TOOL TO PRINT EXECUTION TREE */

DCL I FIXED;

DO I = 1 TO 500 WHILE (XTREE (I).LABEL ^= ''
                     | XTREE (I).CHILD ^= 0
                     | XTREE (I).LINK ^= 0);
   PUT SKIP EDIT ('XTREE_LOC = ',I,
                  'LABEL = ',XTREE (I).LABEL,
                  'CHILD = ',XTREE (I).CHILD,
                  'LINK  = ',XTREE (I).LINK)
                (A,F(5),SKIP,A,A,2(SKIP,A,F(5)));
   END;

END PRINTX;
```

ACT00880
ACT00890
ACT00900
ACT00910
ACT00920
ACT00930
ACT00940
ACT00950
ACT00960
ACT00970
ACT00980
ACT00990
ACT01000
AGT01010
ACT01020
ACT01030
ACT01040
ACT01050
ACT01060
ACT01070
ACT01080

```
                                                                        ACTO1090
                                                                        ACTO1100
                                                                        ACTO1110
                                                                        ACTO1120
                                                                        ACTO1130
PRINTE: PROC;  /* DEBUG TOOL TO PRINT ENTITY SET TABLE */              ACTO1140
                                                                        ACTO1150
   DCL (I,J) FIXED;                                                     ACTO1160
                                                                        ACTO1170
   DO I = 1 TO 12 WHILE (ENTITY (I).VES_FN ^= '');                      ACTO1180
      PUT SKIP EDIT ('ENTITY: ',I)(A,F (2));                            ACTO1190
      PUT SKIP EDIT ('NAME: ',ENTITY (I).NAME)(COL (2),A,A);            ACTO1200
      PUT SKIP EDIT ('VES_FN: ',ENTITY (I).VES_FN)(COL (2),A,A);        ACTO1210
      PUT SKIP EDIT ('WHERE: ',ENTITY (I).WHERE)(COL (2),A,F (4));      ACTO1220
      PUT SKIP EDIT ('N_PARENTS: ',                                     ACTO1230
                     ENTITY (I).N_PARENT (1),                           ACTO1240
                     ENTITY (I).N_PARENT (2))                           ACTO1250
                    (COL (2),A,F (2),X (10),F (2));                     ACTO1260
      DO J = 1 TO 15 WHILE (ENTITY (I).ATTR (J).USES ^='');             ACTO1270
         PUT SKIP EDIT ('ATTRIBUTE: ',J)(COL (5),A,F (2));              ACTO1280
         IF ENTITY (I).ATTR (J).VES_KEY THEN                            ACTO1290
            PUT SKIP EDIT ('VES_KEY')(COL (7),A);                       ACTO1300
         IF ENTITY (I).ATTR (J).CART_KEY THEN                           ACTO1310
            PUT SKIP EDIT ('CART_KEY')(COL (7),A);                      ACTO1320
         PUT SKIP EDIT                                                  ACTO1330
            ('A_PARENT: ',ENTITY (I).ATTR (J).A_PARENT)                 ACTO1340
            (COL (7),A,F (2));                                          ACTO1350
         PUT SKIP EDIT                                                  ACTO1360
            ('USES: ',ENTITY (I).ATTR (J).USES)                         ACTO1370
            (COL (7),A,A);                                              ACTO1380
         END;                                                           ACTO1390
      END;                                                              ACTO1400
                                                                        ACTO1410
END PRINTE;                                                             ACTO1420
                                                                        ACTO1430
```

```
                                                                        ACTO1440
                                                                        ACTO1450
                                                                        ACTO1460
                                                                        ACTO1470
                                                                        ACTO1480
PRINTR: PROC; /* DEBUG TOOL TO PRONT REVISED ENTITY SET TABLE */       ACTO1490
                                                                        ACTO1500
DCL (I,J) FIXED,                                                        ACTO1510
    RP PTR;                                                             ACTO1520
                                                                        ACTO1530
RP = RETEP;                                                             ACTO1540
                                                                        ACTO1550
DO WHILE (RP ^= NULL ());                                               ACTO1560
   PUT SKIP EDIT                                                        ACTO1570
      ('ENT_NAME: ',RP -> RETE_ARG.ENT.NAME,                            ACTO1580
       'ENT_DEPTH: ',RP -> RETE_ARG.ENT.DEPTH)                          ACTO1590
      (A,A,SKIP,A,F (5));                                               ACTO1600
   DO I = 1 TO RP -> RETE_ARG.NUM_ATTR;                                 ACTO1610
      PUT SKIP EDIT ('ATTRIBUTE: ',I)(A,F (3));                         ACTO1620
      IF RP -> RETE_ARG.ENT.ATTR (I).SING_OCC THEN                      ACTO1630
         PUT SKIP EDIT ('SING_OCC')(COL (5),A);                         ACTO1640
      PUT SKIP EDIT                                                     ACTO1650
         ('A_PARENT: ',RP -> RETE_ARG.ENT.ATTR (I).A_PARENT,            ACTO1660
          'USES: ',RP -> RETE_ARG.ENT.ATTR (I).USES)                    ACTO1670
         (COL (5),A,A,SKIP,COL (5),A,A);                                ACTO1680
      END;                                                              ACTO1690
   DO I = 1 TO RP -> RETE_ARG.NUM_COND;                                 ACTO1700
      PUT SKIP EDIT                                                     ACTO1710
         ('COND: ',I,                                                   ACTO1720
          'ATTRREF: ',RP -> RETE_ARG.COND (I).ATTRREF)                  ACTO1730
         (A,F (3),SKIP,COL (2),A,F (3));                                ACTO1740
      IF RP -> RETE_ARG.COND (I).NEG THEN                               ACTO1750
         PUT SKIP EDIT ('NEGATE')(COL (2),A);                           ACTO1760
      PUT SKIP EDIT ('REL: ',RP -> RETE_ARG.COND (I).REL)(COL (2),A,A);ACTO1770
      DO J = 1 TO 10 WHILE (RP -> RETE_ARG.COND (I).CDATA (J) ^= '');   ACTO1780
         PUT SKIP EDIT                                                  ACTO1790
            (J,RP -> RETE_ARG.COND (I).CDATA (J))                       ACTO1800
            (COL (5),F (3),X (1),A);                                    ACTO1810
         END;                                                           ACTO1820
      END;                                                              ACTO1830
   RP = RP -> RETE_ARG.CTL_INFO.PTR;                                    ACTO1840
   END;                                                                 ACTO1850
                                                                        ACTO1860
END PRINTR;                                                             ACTO1870
                                                                        ACTO1880
```

```
GETS: PROC (LIST,TERM_ITEM) RETURNS (CHAR (30) VAR);

DCL LIST CHAR (*) VAR,
    TERM_ITEM CHAR (1),
    RTN_LIST CHAR (30) VAR,
    I FIXED;

I = INDEX (LIST,TERM_ITEM);
IF I = 0 THEN DO;
   RTN_LIST = LIST;
   LIST = '';
   END;
ELSE DO;
   RTN_LIST = SUBSTR (LIST,1,I - 1);
   LIST = SUBSTR (LIST,I + 1);
   END;
RETURN (RTN_LIST);

END GETS;
```

```
ACT01890
ACT01900
ACT01910
ACT01920
ACT01930
ACT01940
ACT01950
ACT01960
ACT01970
ACT01980
ACT01990
ACT02000
ACT02010
ACT02020
ACT02030
ACT02040
ACT02050
ACT02060
ACT02070
ACT02080
ACT02090
ACT02100
ACT02110
ACT02120
ACT02130
```

```
END ACTCRD;                                                    ACTO2140
```

```
                                                                    TKN00010
                                                                    TKN00020
                                                                    TKN00030
                                                                    TKN00040
                                                                    TKN00050
/* THIS PROCEDURE TOKENIZES INPUT RETRIEVE STATEMENTS;              TKN00060
   EXECUTES DEFINE, ADHOC, AND LISTDEF STATEMENTS, AND              TKN00070
   REPLACES VIRTUAL DEFINITION NAMES BY THEIR CORRESPONDING         TKN00080
   VIRTUAL DEFINITIONS.                                    */       TKN00090
TKNIZE: PROC (UNIT, TKLSPTR, DICTION,GO);                           TKN00100
%INCLUDE DICTION;                                                   TKN00110
%INCLUDE TOKEN;                                                     TKN00120
DCL DCTNRY ENTRY (,CHAR (160) VAR)                                  TKN00130
           EXTERNAL RETURNS (CHAR (160) VAR);                       TKN00140
DCL UNIT CHAR (2000) VAR;                                           TKN00150
DCL GO BIT (1);                                                     TKN00160
DCL TKLSPTR PTR;                                                    TKN00170
DCL WORD CHAR (80) VAR;                                             TKN00180
DCL PLIST1(3) CHAR(8) INIT ('DVHUTL','TEST',HIGH(8)),               TKN00190
    PLIST2(3) CHAR(8) INIT ('DVHUTL','CLEAR',HIGH(8)),              TKN00200
    (HIGH, PLIRETC) BUILTIN,                                        TKN00210
    RETCODE FIXED BINARY (31,0),                                    TKN00220
    CMSCMD EXTERNAL ENTRY OPTIONS (ASSEMBLER INTER);                TKN00230
DCL GARBAGE CHAR (2000) VAR ;                                       TKN00240
DCL KIND FIXED INIT (0);                                            TKN00250
CALL CMSCMD (PLIST2,RETCODE);                                       TKN00260
WORD = NXTKSTR (UNIT,KIND);                                         TKN00270
SELECT (WORD);                                                      TKN00280
   WHEN ('DEFINE','DEF','ADHOC')                                    TKN00290
      CALL DEF (UNIT,TKLSPTR,KIND,WORD);                            TKN00300
   WHEN (';') DO;                                                   TKN00310
      UNIT = '' ;                                                   TKN00320
      GO = '0'B;                                                    TKN00330
      RETURN;                                                       TKN00340
      END;                                                          TKN00350
   WHEN ('RETRIEVE','RET')                                          TKN00360
      UNIT = 'RETRIEVE ' || UNIT ;                                  TKN00370
   WHEN ('LISTDEF')                                                 TKN00380
      CALL LISTDEF (UNIT,TKLSPTR);                                  TKN00390
   OTHERWISE DO;                                                    TKN00400
      CALL MSG ('UNRECOGNIZED COMMAND-- ' || WORD);                 TKN00410
      UNIT = '' ;                                                   TKN00420
      TKLSPTR = NULL();                                             TKN00430
      RETURN;                                                       TKN00440
      END;                                                          TKN00450
   END;                                                             TKN00460
                                                                    TKN00470
CALL BDTKCHN (UNIT,TKLSPTR,WORD,KIND);                              TKN00480
                                                                    TKN00490
                                                                    TKN00500
```

```
GETS: PROC (LIST,TERM_ITEM) RETURNS (CHAR (1000) VAR);
DCL LIST CHAR (*) VAR,
    TERM_ITEM CHAR (1),
    RTN_LIST CHAR (1000) VAR,
    I FIXED;

I = INDEX (LIST,TERM_ITEM);
IF I = 0 THEN DO;
   RTN_LIST = LIST;
   LIST = '';
   END;
ELSE DO;
   RTN_LIST = SUBSTR (LIST,1,I - 1);
   LIST = SUBSTR (LIST,I + 1);
   END;
RETURN (RTN_LIST);
END GETS;
```

```
TKN00510
TKN00520
TKN00530
TKN00540
TKN00550
TKN00560
TKN00570
TKN00580
TKN00590
TKN00600
TKN00610
TKN00620
TKN00630
TKN00640
TKN00650
TKN00660
TKN00670
TKN00680
TKN00690
TKN00700
TKN00710
TKN00720
TKN00730
TKN00740
```

```
NXTKSTR: PROC (UNIT,KIND) RETURNS (CHAR (80) VAR);                    TKN00750
DCL UNIT CHAR (2000) VAR;                                             TKN00760
DCL SYMBOL CHAR (1);                                                  TKN00770
DCL KIND FIXED;                                                       TKN00780
KIND = 0;                                                             TKN00790
IF UNIT = '' THEN RETURN ('');                                        TKN00800
DO WHILE ('1'B);                                                      TKN00810
   CALL RMVFBLKS (UNIT);                                              TKN00820
   IF UNIT = '' THEN RETURN ('');                                     TKN00830
   IF INDEX ('\@',SUBSTR (UNIT,1,1)) ^= 0 THEN                        TKN00840
      DO WHILE (INDEX ('\@',SUBSTR (UNIT,1,1)) ^= 0);                 TKN00850
         SYMBOL = SUBSTR (UNIT,1,1);                                  TKN00860
         UNIT = SUBSTR (UNIT,2);                                      TKN00870
         IF SYMBOL = '\' THEN                                         TKN00880
            GARBAGE = GETS (UNIT,SYMBOL);                             TKN00890
         ELSE                                                         TKN00900
            GARBAGE = GETS (UNIT,':');                                TKN00910
         END;                                                         TKN00920
   IF SUBSTR (UNIT,1,1) ^= ' ' THEN                                   TKN00930
      RETURN (TOK1 (UNIT,KIND));                                      TKN00940
   END;                                                               TKN00950
                                                                      TKN00960
                                                                      TKN00970
                                                                      TKN00980
                                                                      TKN00990
                                                                      TKN01000
                                                                      TKN01010
```

```
                                                                    TKNO1020
                                                                    TKNO1030
                                                                    TKNO1040
                                                                    TKNO1050
                                                                    TKNO1060
RMVFBLKS: PROC (STRING);                /* REMOVE FRONT BLANKS */   TKNO1070
DCL STRING CHAR (2000) VAR;                                         TKNO1080
DCL I FIXED;                                                        TKNO1090
IF STRING = '' THEN RETURN;                                         TKNO1100
DO I = 1 TO LENGTH (STRING);                                        TKNO1110
   IF SUBSTR (STRING,I,1) ^= ' ' THEN DO;                           TKNO1120
      STRING = SUBSTR (STRING, I);                                  TKNO1130
      RETURN;                                                       TKNO1140
      END;                                                          TKNO1150
   END;                                                             TKNO1160
STRING = '' ;                                                       TKNO1170
RETURN;                                                             TKNO1180
END RMVFBLKS;                                                       TKNO1190
```

```
TOK1: PROC (UNIT,KIND) RETURNS (CHAR (80) VAR);                          TKNO1200
DCL (UNIT,UNITCLAS) CHAR (2000) VAR;                                     TKNO1210
DCL (KIND,I,J) FIXED;                                                    TKNO1220
DCL TKSTR CHAR (80) VAR;                                                 TKNO1230
DCL (CLASS,PRECLASS) CHAR (1);                                           TKNO1240
IF UNIT = '' THEN RETURN ('');                                           TKNO1250
UNITCLAS = TRANSLATE (UNIT,                                              TKNO1260
'AAAAAAAAAAAAAAAAAAAAAAAAAANNNNNNNNNNDOOOOOOBBBBBBBBBBBQMMMMMMMMMS',   TKNO1270
'ABCDEFGHIJKLMNOPQRSTUVWXYZO123456789.+-/*¦!^=><  .()~{};'''"?¢_&`#$¦%');TKNO1280
TKSTR = '' ;                                                             TKNO1290
PRECLASS = 'B';                                                          TKNO1300
DO I = 1 TO LENGTH (UNIT);                                               TKNO1310
   CLASS = SUBSTR (UNITCLAS,I,1);                                        TKNO1320
   SELECT (PRECLASS);                                                    TKNO1330
      WHEN ('B','M','O') DO;                                             TKNO1340
         TKSTR = SUBSTR (UNIT,I,1);                                      TKNO1350
         IF CLASS = 'B' ¦ CLASS = 'M' ¦ CLASS = 'O' THEN                 TKNO1360
            IF I = LENGTH (UNIT) THEN DO;                                TKNO1370
               UNIT = '' ;                                               TKNO1380
               RETURN (TKSTR);                                           TKNO1390
               END;                                                      TKNO1400
            ELSE DO;                                                     TKNO1410
               UNIT = SUBSTR (UNIT,I+1);                                 TKNO1420
               RETURN (TKSTR);                                           TKNO1430
               END;                                                      TKNO1440
         PRECLASS = CLASS;                                               TKNO1450
         IF I = LENGTH (UNIT) THEN DO;                                   TKNO1460
            UNIT = '' ;                                                  TKNO1470
            IF CLASS = 'N' THEN                                          TKNO1480
               KIND = 1;                                                 TKNO1490
            RETURN (TKSTR);                                              TKNO1500
            END;                                                         TKNO1510
         END;                                                            TKNO1520
      WHEN ('A')                                                         TKNO1530
         IF INDEX ('ANOM',CLASS) ^= 0 THEN DO;                           TKNO1540
            TKSTR = TKSTR ¦¦ SUBSTR (UNIT,I,1);                          TKNO1550
            IF I = LENGTH (UNIT) THEN DO;                                TKNO1560
               UNIT = '' ;                                               TKNO1570
               RETURN (TKSTR);                                           TKNO1580
               END;                                                      TKNO1590
            END;                                                         TKNO1600
         ELSE DO;                                                        TKNO1610
            UNIT = SUBSTR (UNIT,I);                                      TKNO1620
            RETURN (TKSTR);                                              TKNO1630
            END;                                                         TKNO1640
      WHEN ('N')                                                         TKNO1650
         IF INDEX ('NO',CLASS) ^= 0 THEN DO;                             TKNO1660
            TKSTR = TKSTR ¦¦ SUBSTR (UNIT,I,1);                          TKNO1670
            IF I = LENGTH (UNIT) THEN DO;                                TKNO1680
               UNIT = '' ;                                               TKNO1690
               KIND = 1;                                                 TKNO1700
               RETURN (TKSTR);                                           TKNO1710
               END;                                                      TKNO1720
```

```
                END;                                                    TKNO1780
             ELSE DO;                                                   TKNO1790
                KIND = 1;                                               TKNO1800
                UNIT = SUBSTR (UNIT,I);                                 TKNO1810
                RETURN (TKSTR);                                         TKNO1820
                END;                                                    TKNO1830
          WHEN ('D')                                                    TKNO1840
             IF CLASS = 'N' THEN DO;                                    TKNO1850
                TKSTR = TKSTR || SUBSTR (UNIT,I,1);                     TKNO1860
                IF I = LENGTH (UNIT) THEN DO;                           TKNO1870
                   KIND = 1;                                            TKNO1880
                   UNIT = '' ;                                          TKNO1890
                   RETURN (TKSTR);                                      TKNO1900
                   END;                                                 TKNO1910
                END;                                                    TKNO1920
             ELSE DO;                                                   TKNO1930
                UNIT = SUBSTR (UNIT,I);                                 TKNO1940
                KIND = 1;                                               TKNO1950
                RETURN (TKSTR);                                         TKNO1960
                END;                                                    TKNO1970
          WHEN ('Q') DO;                                                TKNO1980
             J = INDEX (SUBSTR (UNITCLAS,2),'Q');                       TKNO1990
             IF J > 0 THEN                                              TKNO2000
                TKSTR = SUBSTR (UNIT,2,J-1);                            TKNO2010
             ELSE                                                       TKNO2020
                TKSTR = UNIT;                                           TKNO2030
             IF J = LENGTH (UNIT) - 1 THEN                              TKNO2040
                UNIT = '' ;                                             TKNO2050
             ELSE                                                       TKNO2060
                UNIT = SUBSTR (UNIT,J+2);                               TKNO2070
             KIND = 2;                                                  TKNO2080
             RETURN (TKSTR);                                            TKNO2090
             END;                                                       TKNO2100
          WHEN ('S') DO;                                                TKNO2110
             TKSTR = TKSTR || SUBSTR (UNIT,I,1) ;                       TKNO2120
             IF I = LENGTH (UNIT) THEN                                  TKNO2130
                UNIT = '' ;                                             TKNO2140
             ELSE                                                       TKNO2150
                UNIT = SUBSTR (UNIT,I+1);                               TKNO2160
             RETURN (TKSTR);                                            TKNO2170
             END;                                                       TKNO2180
          OTHERWISE;                                                    TKNO2190
          END;                                                          TKNO2200
       END;                                                             TKNO2210
    END TOK1;                                                           TKNO2220
    END NXTKSTR;                                                        TKNO2230
```

```
/* THIS PROCEDURE DEFINES AND REMOVES DEFINITIONS FROM
   ADHOC AND/OR PERMANENT DICTIONARIES                    */
DEF: PROC (UNIT,TKLSPTR,KIND,WORD);
DCL UNIT CHAR (2000) VAR;
DCL TKLSPTR PTR;
DCL KIND FIXED;
DCL (WORD,WORDO,VNAME,VNAME1) CHAR (80) VAR;
DCL WORD1 CHAR (160) VAR;
DCL DEFMSG CHAR (160) VAR;
IF WORD = 'DEF' THEN WORD = 'DEFINE';
WORDO = WORD;
VNAME = NXTKSTR (UNIT,KIND);
VNAME1 = TRANSLATE (VNAME,'11111111111111111111111',
                          '0123456789@#$%&_¦¢"{}?,''');
IF SUBSTR (VNAME1,1,1) = '1' THEN DO;
   CALL MSG ('@' || VNAME || '@ DEFINITION: *ERROR* INVALID NAME');
   UNIT = '';
   TKLSPTR = NULL();
   RETURN;
   END;
WORD = NXTKSTR (UNIT,KIND);
IF WORD ^= 'AS' THEN
   IF ^ (WORD = 'REMOVE' ¦ WORD = 'REM') THEN DO;
      UNIT = '';
      TKLSPTR = NULL();
      CALL MSG ('MISSING "AS" IN DEFINE OR ADHOC STMT');
      RETURN;
      END;
   ELSE DO;
      IF WORDO = 'DEFINE' THEN
         DEFMSG = DCTNRY (DICTION,'REMOVE,'||VNAME);
      ELSE
         DEFMSG = DCTNRY (DICTION,'ADHREMOVE,'||VNAME);
      UNIT = '' ;
      CALL MSG ('@' || VNAME || '@ REMOVE: ' || GETS (DEFMSG,','));
      END;
ELSE DO;
   TKLSPTR = NULL();
   WORD1 = GETS (UNIT,';');
   IF WORDO = 'DEFINE' THEN
      WORD1 = 'SAVE,' || VNAME || ',' || WORD1 ;
   ELSE
      WORD1 = 'ADHSAVE,' ||  VNAME || ',' || WORD1 ;
   DEFMSG = DCTNRY (DICTION,WORD1);
   CALL MSG ('@' || VNAME || '@ DEFINITION: ' || GETS (DEFMSG,','));
   END;
GO = '0'B;
END DEF;
```

TKN02290
TKN02300
TKN02310
TKN02320
TKN02330
TKN02340
TKN02350
TKN02360
TKN02370
TKN02380
TKN02390
TKN02400
TKN02410
TKN02420
TKN02430
TKN02440
TKN02450
TKN02460
TKN02470
TKN02480
TKN02490
TKN02500
TKN02510
TKN02520
TKN02530
TKN02540
TKN02550
TKN02560
TKN02570
TKN02580
TKN02590
TKN02600
TKN02610
TKN02620
TKN02630
TKN02640
TKN02650
TKN02660
TKN02670
TKN02680
TKN02690
TKN02700
TKN02710
TKN02720
TKN02730
TKN02740
TKN02750
TKN02760
TKN02770
TKN02780
TKN02790
TKN02800
TKN02810
TKN02820

```
                                                                        TKN02830
                                                                        TKN02840
                                                                        TKN02850
                                                                        TKN02860
                                                                        TKN02870
BDTKCHN: PROC (UNIT,TKLSPTR,WORD,KIND); /* BUILDS TOKEN CHAIN */       TKN02880
DCL UNIT CHAR (2000) VAR;                                               TKN02890
DCL (TKLSPTR,P,TAIL) PTR;                                               TKN02900
DCL WORD CHAR (80) VAR;                                                 TKN02910
DCL (FINDMSG,CPFMSG,SUBSTITUTE) CHAR (160) VAR;                         TKN02920
DCL KIND FIXED;                                                         TKN02930
DCL WORD1 CHAR (160) VAR;                                               TKN02940
DO WHILE (UNIT ^= '');                                                  TKN02950
   WORD = NXTKSTR (UNIT,KIND);                                          TKN02960
   IF KIND ^= 0 THEN DO;                                                TKN02970
      ALLOCATE TOKEN SET (P);                                           TKN02980
      IF KIND = 1 THEN DO;                                              TKN02990
         P -> TOKEN.CLASS = ':A' ;                                      TKN03000
         P -> TOKEN.ITEM = CHAR (FIXED (WORD));                         TKN03010
         END;                                                           TKN03020
      ELSE DO;                                                          TKN03030
         P -> TOKEN.CLASS = ':S' ;                                      TKN03040
         P -> TOKEN.ITEM = WORD;                                        TKN03050
         END;                                                           TKN03060
      P -> TOKEN.NEXT = NULL ();                                        TKN03070
      IF TKLSPTR = NULL () THEN                                         TKN03080
         TKLSPTR = P;                                                   TKN03090
      ELSE                                                              TKN03100
         TAIL -> TOKEN.NEXT = P;                                        TKN03110
      TAIL = P;                                                         TKN03120
      END;                                                              TKN03130
   ELSE DO;                                                             TKN03140
      WORD1 = 'FIND.' || WORD ;                                         TKN03150
      FINDMSG = DCTNRY (DICTION,WORD1);                                 TKN03160
      CPFMSG = FINDMSG;                                                 TKN03170
      SUBSTITUTE = GETS (CPFMSG,',');                                   TKN03180
      IF CPFMSG ^= ':V' THEN DO;                                        TKN03190
         IF SUBSTR (SUBSTITUTE,1,1) = '1' THEN DO;                      TKN03200
            CALL MSG (WORD || ': ' || SUBSTR (SUBSTITUTE,2));           TKN03210
            UNIT = '';                                                  TKN03220
            TKLSPTR = NULL();                                           TKN03230
            RETURN;                                                     TKN03240
            END;                                                        TKN03250
         ALLOCATE TOKEN SET (P);                                        TKN03260
         P -> TOKEN.ITEM = GETS (FINDMSG,',');                          TKN03270
         P -> TOKEN.CLASS = FINDMSG;                                    TKN03280
         P -> TOKEN.NEXT = NULL ();                                     TKN03290
         IF TKLSPTR = NULL() THEN                                       TKN03300
            TKLSPTR = P;                                                TKN03310
         ELSE                                                           TKN03320
            TAIL -> TOKEN.NEXT = P;                                     TKN03330
         TAIL = P;                                                      TKN03340
         END;                                                           TKN03350
      ELSE                                                              TKN03360
         UNIT = SUBSTITUTE || ' ' || UNIT ;                             TKN03370
      END;                                                              TKN03380
   END;                                                                 TKN03390
END BDTKCHN;                                                            TKN03400
```

```
                                                                    TKNO3410
                                                                    TKNO3420
                                                                    TKNO3430
                                                                    TKNO3440
                                                                    TKNO3450
/* THIS PROCEDURE OUTPUTS ARBITRRY MESSAGES AND WAITS              TKNO3460
   FOR THE USER TO TYPE THE "ENTER" KEY TO CONTINUE */              TKNO3470
MSG: PROC(LINE);                                                    TKNO3480
DCL LINE CHAR (*) VAR;                                              TKNO3490
CALL CMSCMD (PLIST2,RETCODE);                                       TKNO3500
PUT SKIP EDIT (LINE) (A);                                           TKNO3510
PUT SKIP EDIT ('TYPE "ENTER" KEY TO CONTINUE') (A);                 TKNO3520
GET EDIT (GARBAGE) (A(80));                                         TKNO3530
GO = '0'B;                                                          TKNO3540
END MSG;                                                            TKNO3550
```

```
                                                                        TKN03560
                                                                        TKN03570
                                                                        TKN03580
                                                                        TKN03590
                                                                        TKN03600
/* THIS PROCEDURE OUTPUTS THE STORED DEFINITIONS */                     TKN03610
LISTDEF: PROC (UNIT,TKLSPTR);                                           TKN03620
DCL UNIT CHAR (2000) VAR;                                               TKN03630
DCL (DEFMSG,DEFMSG1) CHAR (160) VAR;                                    TKN03640
DCL WORD CHAR (80) VAR;                                                 TKN03650
DCL TKLSPTR PTR;                                                        TKN03660
DCL KIND FIXED INIT (0);                                                TKN03670
UNIT = SUBSTR (UNIT,1,INDEX (UNIT,';')-1);                              TKN03680
WORD = NXTKSTR (UNIT,KIND);                                             TKN03690
UNIT = '';                                                              TKN03700
TKLSPTR = NULL();                                                       TKN03710
IF WORD = '' | KIND = 1 | KIND = 2 THEN DO;                             TKN03720
   CALL MSG ('*ERROR* INVALID ARGUMENT TO "LISTDEF" STATEMENT');        TKN03730
   RETURN;                                                              TKN03740
   END;                                                                 TKN03750
DEFMSG = DCTNRY (DICTION,'FIND,'||WORD);                                TKN03760
DEFMSG1 = GETS (DEFMSG,',');                                            TKN03770
IF DEFMSG ^= ':V' THEN DO;                                              TKN03780
   CALL MSG ('*ERROR* NO DEFINITION STORED FOR @' || WORD || '@');      TKN03790
   RETURN;                                                              TKN03800
   END;                                                                 TKN03810
CALL CMSCMD (PLIST2,RETCODE);                                           TKN03820
PUT SKIP EDIT                                                           TKN03830
   ('DEFINITION OF "' || WORD || '": ') (A(80));                        TKN03840
CALL DEFDSPLY (DEFMSG1);                                                TKN03850
GET EDIT (GARBAGE) (A(80));                                             TKN03860
GO = 'O'B;                                                              TKN03870
RETURN;                                                                 TKN03880
```

```
/* THIS PROCEDURE IS CALLED BY LISTDEF TO TRANSLATE                    TKN03890
   PREVIOUSLY TRANSLATED CHARACTERS WITHIN THE VIRTUAL                 TKN03900
   DEFINITIONS, AND THEN TO DISPLAY THE STORED                         TKN03910
   VIRTUAL DEFINITIONS.                                 */             TKN03920
DEFDSPLY: PROC (DFNTN);                                                TKN03930
DCL DFNTN CHAR (160) VAR;                                              TKN03940
DCL LINE CHAR (160) VAR;                                               TKN03950
DCL I FIXED;                                                           TKN03960
DCL SPACES FIXED INIT (0);                                             TKN03970
DCL SPACECH CHAR (2) VAR;                                              TKN03980
DO WHILE (DFNTN ^= '');                                                TKN03990
   LINE = GETS (DFNTN,'¢');                                            TKN04000
   IF LINE ^= '' THEN DO;                                              TKN04010
      CALL REPLACE (LINE,'%1','''''');                                 TKN04020
      CALL REPLACE (LINE,'%2',';');                                    TKN04030
      CALL REPLACE (LINE,'%3',':');                                    TKN04040
      CALL REPLACE (LINE,'%5','¢');                                    TKN04050
      CALL REPLACE (LINE,'%4','%');                                    TKN04060
      IF SPACES ^= 0 THEN                                              TKN04070
         DO I = 1 TO SPACES;                                           TKN04080
            LINE = ' ' || LINE;                                        TKN04090
            END;                                                       TKN04100
      PUT EDIT (LINE) (COL(1),A(80));                                  TKN04110
      IF DFNTN ^= '' THEN                                              TKN04120
         DFNTN = '¢' || DFNTN;                                         TKN04130
      END;                                                             TKN04140
   ELSE DO;                                                            TKN04150
      SPACECH = GETS (DFNTN,':');                                      TKN04160
      SPACES = FIXED (SPACECH);                                        TKN04170
      END;                                                             TKN04180
   END;                                                                TKN04190
END DEFDSPLY;                                                          TKN04200
END LISTDEF;                                                           TKN04210
                                                                       TKN04220
                                                                       TKN04230
                                                                       TKN04240
                                                                       TKN04250
                                                                       TKN04260
                                                                       TKN04270
```

```
                                                                    TKN04280
                                                                    TKN04290
                                                                    TKN04300
                                                                    TKN04310
                                                                    TKN04320
   /* THIS PROCEDURE REPLACES CERTAIN CHARACTERS BY OTHER           TKN04330
      PRESCRIBED CHARACTERS                                  */     TKN04340
   REPLACE: PROC (RLINE,FR_SYM,TO_SYM);                             TKN04350
   DCL (RLINE,REP_LINE) CHAR (160) VAR,                             TKN04360
       FR_SYM CHAR (2) VAR,                                         TKN04370
       TO_SYM CHAR (2) VAR,                                         TKN04380
       I FIXED;                                                     TKN04390
                                                                    TKN04400
   REP_LINE = '';                                                   TKN04410
   I = INDEX (RLINE,FR_SYM);                                        TKN04420
   DO WHILE (I ^= 0);                                               TKN04430
      IF LENGTH (FR_SYM) = 1 THEN DO;                               TKN04440
         REP_LINE= REP_LINE || SUBSTR (RLINE,1,I - 1) || TO_SYM;    TKN04450
         RLINE = SUBSTR (RLINE,I + 1);                              TKN04460
         I = INDEX (RLINE,FR_SYM);                                  TKN04470
         END;                                                       TKN04480
      ELSE DO;                                                      TKN04490
         REP_LINE = REP_LINE || SUBSTR (RLINE,1,I-1) || TO_SYM;     TKN04500
         RLINE = SUBSTR (RLINE,I+2);                                TKN04510
         I = INDEX (RLINE,FR_SYM);                                  TKN04520
         END;                                                       TKN04530
      END;                                                          TKN04540
         RLINE = REP_LINE || RLINE;                                 TKN04550
                                                                    TKN04560
   END REPLACE;                                                     TKN04570
                                                                    TKN04580
                                                                    TKN04590
   FINISHED:                                                        TKN04600
   END TKNIZE;                                                      TKN04610
```

DATA STRUCTURES

```
DCL 1 MACH,                                                    ALL00020
    2 STATE_MAP (200) FIXED,                                   ALL00030
    2 MATCH (500) CHAR (30) VAR,                               ALL00040
    2 ACTION (500) CHAR (40) VAR,                              ALL00050
    2 NEXT_STATE (500) FIXED;                                  ALL00060
DCL 1 TOKEN BASED,                                             ALL00070
    2 ITEM CHAR (30) VAR,                                      ALL00080
    2 CLASS CHAR (10) VAR,                                     ALL00090
    2 NEXT PTR;                                                ALL00100
DCL 1 XTREE (1000),                                            ALL00110
    2 LABEL CHAR (30) VAR,                                     ALL00120
    2 CHILD FIXED,                                             ALL00130
    2 LINK FIXED;                                              ALL00140
DCL 1 ATTRIB BASED,                                            ALL00150
    2 LEVEL FIXED,                                             ALL00160
    2 ITEM CHAR (50) VAR,                                      ALL00170
    2 NEXT PTR;                                                ALL00180
DCL 1 ENTITY (12),                                             ALL00190
      2 NAME CHAR (30) VAR,                                    ALL00200
      2 DEPTH FIXED,                                           ALL00210
      2 VES_FN CHAR (2) VAR,                                   ALL00220
      2 WHERE FIXED,                                           ALL00230
      2 N_PARENT (2) FIXED,                                    ALL00240
      2 VES_PAR PTR,                                           ALL00250
      2 ATTR (15),                                             ALL00260
        3 VES_KEY BIT (1),                                     ALL00270
        3 CART_KEY BIT (1),                                    ALL00280
        3 SING_OCC BIT (1),                                    ALL00290
        3 A_PARENT FIXED,                                      ALL00300
        3 USES CHAR (30) VAR,                                  ALL00310
        3 LIST PTR;                                            ALL00320
DCL 1 N_MAP (2) BASED,                                         ALL00330
      2 NUM (15) FIXED;                                        ALL00340
```

```
                                                                    ALL00410
                                                                    ALL00420
                                                                    ALL00430
                                                                    ALL00440
                                                                    ALL00450
                                                                    ALL00460
DCL 1 DICTION,                                                      ALL00470
      2 CHECK,                                                      ALL00480
        3 LIMIT FIXED,                                              ALL00490
        3 LABEL (50) CHAR (50) VAR,                                 ALL00500
        3 TIMES (50) FIXED,                                         ALL00510
      2 MAIN,                                                       ALL00520
        3 LIMIT FIXED,                                              ALL00530
        3 FROM (100) CHAR (50) VAR,                                 ALL00540
        3 TO (100) CHAR (160) VAR,                                  ALL00550
      2 ADHOC,                                                      ALL00560
        3 LIMIT FIXED,                                              ALL00570
        3 FROM (50) CHAR (50) VAR,                                  ALL00580
        3 TO (50) CHAR (160) VAR;                                   ALL00590
DCL 1 ENTCOND (12,12),                                              ALL00600
    2 ATTRREF FIXED,                                                ALL00610
    2 NEG BIT (1),                                                  ALL00620
    2 REL CHAR (1) VAR,                                             ALL00630
    2 CDATA (10) CHAR (30) VAR;                                     ALL00640
DOBYXGRGGE (20) CHAR (80) VAR;                                      ALL00660

DCL 1 RETE_ARG BASED,  /* WILL ALSO BE RETE_RTN */                  ALL00670
      2 CTL_INFO,                                                   ALL00680
        3 LEN FIXED BIN (15),                                       ALL00690
        3 CBTP FIXED BIN (15) INIT (21),                            ALL00700
        3 PTR PTR,                                                  ALL00710
      2 NUM_ATTR FIXED,                                             ALL00720
      2 NUM_COND FIXED,   /* NUMBER OF CONDITIONS */                ALL00730
      2 ENT,                                                        ALL00740
        3 NAME CHAR (30) VAR,  /* ENTITY SET NAME */                ALL00750
        3 DEPTH FIXED,     /* FILLED IN WHEN RETURNED */            ALL00760
        3 ATTR (NATTR REFER (RETE_ARG.NUM_ATTR)),                   ALL00770
          4 SING_OCC BIT (1), /* IF SINGLE OCCUR THEN SET */        ALL00780
          4 A_PARENT FIXED, /* PARENT NUMBER */                     ALL00790
          4 USES CHAR(30) VAR, /* ATTRIBUTE NAME */                 ALL00800
          4 LIST PTR, /* POINT TO LIST OF OCC OF THIS ATTR IF ANY */ ALL00810
      2 COND (NCOND REFER (RETE_ARG.NUM_COND)),                     ALL00820
        3 ATTRREF FIXED, /* POINTER TO ATTR IN ATTR ARRAY ABOVE */  ALL00830
        3 NEG BIT (1),     /*DOES A NEGATION ON RELATOR*/           ALL00840
        3 REL CHAR (1), /* '=', '<', '>' */                         ALL00850
        3 CDATA (10) CHAR (30) VAR;  /*UP TO 10 "MULTI" ITEMS*/     ALL00860
```

```
DCL 1 RETE_RTN1 BASED,  /* USED WHEN RETURNED */
      2 CTL,
        3 LEN FIXED BIN(31),
        3 CBTP FIXED BIN (31) INIT (46),
        3 PTR PTR,
      2 LEVEL FIXED, /* OCCUR NUMBER */
      2 ITEM CHAR (50) VAR;
```

ALL00870
ALL00880
ALL00890
ALL00900
ALL00910
ALL00920
ALL00930
ALL00940
ALL00950
ALL00960
ALL00970
ALL00980
ALL00990

FINITE-STAE-MACHINE RULES
MATCH - ACTION - NEXT_STATE

STATE NUMBER: 1			
1. 1	RETRIEVE	PUSH,2,I@¦DEL¦PUSH,2,SUBR:2	47
STATE NUMBER: 2			
2. 1	;,,RETRIEVE		0
2. 2	BY	DEL¦PUSH,2,SUBR:2	140
STATE NUMBER: 3			
STATE NUMBER: 11			
11. 1	~	DEL	18
11. 2	(	PUSH,2,I@¦DEL	12
11. 3	:A¦:S	PUSH,1,C@¦DEL	12
11. 4	-	PUSH,2,I@P:1¦DEL	16
11. 5	+	DEL	11
11. 6	:R	PUSH,1,C@¦DEL	14
11. 7	STR	PUSH,2,SUBR:12¦PUSH,2,I@:6¦DEL	22
11. 8	DATE	PUSH,2,I@:0¦GENNODE¦DEL	12
11. 9	:B	PUSH,2,SUBR:12¦PUSH,2,I@:1¦DEL	20
STATE NUMBER: 12			
12. 1	),,(	DEL¦POP,2	12
12. 2	,,-P	GENNODE	12
12. 3	!	PUSH,2,I@:2¦DEL	13
12. 4	@SUMOP,,@SUMOP>	GENNODE	12
12. 5	@SUMOP>,,@CONC	PUSH,2,I@:2¦DEL	13
12. 6	@MULTOP,,@MULTOP>	GENNODE	12
12. 7	@MULTOP,,@SUMOP	PUSH,2,I@:2¦DEL	13
12. 8	@CONC,,@CONC>	GENNODE	12
12. 9	@CONC>	PUSH,2,I@:2¦DEL	13
12.10	,,SUBR		-1
12.11		GENNODE	12
STATE NUMBER: 13			
13. 1	:R	PUSH,1,C@¦DEL	14
13. 2	:A¦:S	PUSH,1,C@¦DEL	12
13. 3	-	PUSH,2,I@P:1¦DEL	16
13. 4	+	DEL	13
13. 5	(	PUSH,2,I@¦DEL	13
13. 6	:B	PUSH,2,I@:1¦DEL	17
STATE NUMBER: 14			
14. 1	(	PUSH,2,I@¦DEL¦INDX,1	15
14. 2			12
STATE NUMBER: 15			
15. 1	:R	PUSH,1,I@¦DEL	19
STATE NUMBER: 16			
16. 1	-	PUSH,2,I@P:1¦DEL	16
16. 2	+	DEL	16
16. 3	:A¦:S	PUSH,1,C@¦DEL	12
16. 4	:R	PUSH,1,C@¦DEL	14

```
                    FINITE-STATE-MACHINE RULES
                    MATCH - ACTION - NEXT_STATE

STATE NUMBER:  17
  17. 1  (                  PUSH,2,I@¦DEL                                  11

STATE NUMBER:  18
  18. 1  :R                 PUSH,1,C@;;C¦DEL                               14

STATE NUMBER:  19
  19. 1  (                  PUSH,2,I@¦DEL¦ADDON,1@¦POP,1                   15
  19. 2  ),,(               ADDON,1@¦POP,1¦DEL¦POP,2                       28

STATE NUMBER:  20
  20. 1  (                  DEL¦PUSH,2,SUBR:21                             11

STATE NUMBER:  21
  21. 1  ,,SUBR                                                           -1
  21. 2  )                  DEL¦GENNODE                                    21

STATE NUMBER:  22
  22. 1  (                  DEL¦PUSH,2,SUBR:23                             11

STATE NUMBER:  23
  23. 1  ,,SUBR                                                           -1
  23. 2  %0                 DEL¦PUSH,2,SUBR:24                             11

STATE NUMBER:  24
  24. 1  ,,SUBR                                                           -1
  24. 2  )                  DEL¦P,1,:-1¦P,1,:-1¦P,1,:-1¦P,1,:-1¦GD         24
  24. 3  %0                 DEL¦P,1,:-1¦P,1,:-2¦P,2,SUBR:26                11
  24. 4  %5                 DEL¦P,1,:-1¦P,1,:-3¦P,2,SUBR:26                11
  24. 5  %3                 DEL¦P,2,SUBR:25                                11

STATE NUMBER:  25
  25. 1  ,,SUBR                                                           -1
  25. 2  )                  DEL¦P,1,:-1¦P,1,:-1¦P,1,:-1¦GD                 25
  25. 3  %0                 DEL¦P,1,:-2¦PUSH,2,SUBR:26                     11
  25. 4  %5                 DEL¦P,1,:-3¦PUSH,2,SUBR:26                     11

STATE NUMBER:  26
  26. 1  ,,SUBR                                                           -1
  26. 2  )                  DEL¦P,1,:-1¦GENNODE                            26
  26. 3  %3                 DEL¦PUSH,2,SUBR:27                             11

STATE NUMBER:  27
  27. 1  ,,SUBR                                                           -1
  27. 2  )                  DEL¦GENNODE                                    27

STATE NUMBER:  28
  28. 1  ),,(               DEL¦POP,2                                      28
  28. 2                                                                    12

STATE NUMBER:  29
```

FINITE-STATE-MACHINE RULES
MATCH - ACTION - NEXT_STATE

```
STATE NUMBER:  30
  30. 1  (                    PUSH,2,I@¦DEL                    31

STATE NUMBER:  31
  31. 1  ^                    PUSH,2,I@:1¦DEL                  36
  31. 2  (                    PUSH,2,I@¦DEL                    32
  31. 3                       PUSH,2,SUBR:33                   11

STATE NUMBER:  32
  32. 1  (                    PUSH,2,I@¦DEL                    32
  32. 2                       PUSH,2,SUBR:33                   11

STATE NUMBER:  33
  33. 1  ^                    PUSH,2,I@:1¦DEL                  37
  33. 2  ).,(                 POP,2¦DEL                        33
  33. 3  =                    PUSH,2,I@:2¦DEL                  38
  33. 4  @REL                 PUSH,2,I@:2¦DEL                  34

STATE NUMBER:  34
  34. 1  (                    PUSH,2,I@¦DEL                    34
  34. 2                       PUSH,2,SUBR:35                   11

STATE NUMBER:  35
  35. 1  ).,(                 POP,2¦DEL                        35
  35. 2  .,@REL               GENNODE                          35
  35. 3  .,^                  GENNODE                          35
  35. 4  .,@CMP               GENNODE                          35
  35. 5  @CMP                 PUSH,2,I@:2¦DEL                  31
  35. 6  .,SUBR                                                -1

STATE NUMBER:  36
  36. 1  ^                    PUSH,2,I@:1¦DEL                  36
  36. 2  (                    PUSH,2,I@¦DEL                    32

STATE NUMBER:  37
  37. 1  ^                    PUSH,2,I@:1¦DEL                  37
  37. 2  @REL                 PUSH,2,I@:2¦DEL                  34

STATE NUMBER:  38
  38. 1  (                    PUSH,2,I@¦DEL                    39
  38. 2                                                        34

STATE NUMBER:  39
  39. 1  :A¦:S                PUSH,1,I@¦DEL                    40
  39. 2                                                        34

STATE NUMBER:  40
  40. 1  ).,(                 DEL¦POP,2¦GENNODE                35
  40. 2  %O                   DEL¦MULX,1                       41
```

FINITE-STATE-MACHINE RULES
MATCH - ACTION - NEXT_STATE

STATE NUMBER: 41		
41. 1 :A¦:S	PUSH,1,I@¦DEL	42
STATE NUMBER: 42		
42. 1),,(	DEL¦POP,2¦ADDON,1@¦POP,1¦GENNODE	35
42. 2 %O	DEL¦ADDON,1@¦POP,1	41
STATE NUMBER: 43		
STATE NUMBER: 47		
47. 1 (	DEL	48
STATE NUMBER: 48		
48. 1 (	PUSH,2,I@¦DEL	54
STATE NUMBER: 49		
49. 1),@VIRT	VIRTX,1,1@¦DEL	52
49. 2)	GENENT¦DEL	50
STATE NUMBER: 50		
50. 1 WHERE	DEL¦PUSH,2,SUBR:51	30
50. 2		52
STATE NUMBER: 51		
51. 1	ATTWHR	52
STATE NUMBER: 52		
52. 1 ;		53
52. 2 BY		53
52. 3)	DEL	53
52. 4 %O	DEL¦VIRTA¦PUSH,2,SUBR:52	48
STATE NUMBER: 53		
53. 1 ,,SUBR		-1
STATE NUMBER: 54		
54. 1 :R	PUSH,1,I@¦DEL	49
54. 2 @VIRT	PUSH,1,I@¦DEL	49
54. 3 (		100
STATE NUMBER: 55		
55. 1 (	PUSH,2,I@¦DEL	56
STATE NUMBER: 56		
56. 1 :R	PUSH,1,I@¦DEL	57
56. 2 @VIRT	PUSH,1,I@¦DEL	57
56. 3 (		112

FINITE-STATE-MACHINE RULES
MATCH - ACTION - NEXT_STATE

STATE NUMBER: 57		
57. 1),@VIRT	VIRTX,1,1@¦DEL	60
57. 2)	GENENT¦DEL	58
STATE NUMBER: 58		
58. 1 WHERE	DEL¦PUSH,2,SUBR:59	30
58. 2		60
STATE NUMBER: 59		
59. 1	ATTWHR	60
STATE NUMBER: 60		70
60. 1 @SETOP		-1
60. 2 .,SUBR	GENENT¦DEL	58
60. 3)		
STATE NUMBER: 61		
61. 1 (	PUSH,2,I@¦DEL	62
STATE NUMBER: 62		
62. 1 :R	PUSH,1,C@¦DEL	63
62. 2 @VIRT	PUSH,1,I@¦DEL	63
62. 3 (		124
STATE NUMBER: 63		
63. 1),@VIRT	VIRTX,1,1@¦DEL	65
63. 2	GENENT¦DEL	64
STATE NUMBER: 64		
64. 1 WHERE	DEL¦PUSH,2,SUBR:65	30
64. 2		66
STATE NUMBER: 65		
65. 1	ATTWHR	66
STATE NUMBER: 66		-1
66. 1 .,SUBR	GENENT¦DEL	64
66. 2)		
STATE NUMBER: 67		

FINITE-STATE-MACHINE RULES
MATCH - ACTION - NEXT_STATE

STATE NUMBER: 70		
70. 1 CS	PUSH,2,I⊕¦DEL	71
70. 2		80
STATE NUMBER: 71		
71. 1 (	PUSH,2,I⊕¦DEL	72
STATE NUMBER: 72		
72. 1 -	DEL	73
STATE NUMBER: 73		
73. 1 :R	PUSH,1,C⊕;V;C¦DEL	74
STATE NUMBER: 74		
74. 1 (	PUSH,2,IND(¦DEL¦INDX,1	75
74. 2		78
STATE NUMBER: 75		
75. 1 :R	PUSH,1,I⊕¦DEL	76
STATE NUMBER: 76		
76. 1 (	PUSH,2,IND(¦DEL¦ADDON,1⊕¦POP,1	75
76. 2),,IND(	ADDON,1⊕¦POP,1¦DEL¦POP,2	77
STATE NUMBER: 77		
77. 1),,IND(	DEL¦POP,2	77
77. 2		78
STATE NUMBER: 78		
78. 1 %O	PUSH,2,I⊕¦DEL	90
STATE NUMBER: 79		
STATE NUMBER: 80		
80. 1	PUSH,2,I⊕¦DEL	81
STATE NUMBER: 81		
81. 1 (	PUSH,2,I⊕¦DEL	82
STATE NUMBER: 82		
82. 1 :R	PUSH,1,C⊕;V¦DEL	83
STATE NUMBER: 83		
83. 1)	PUSH,2,I⊕¦DEL	61
83. 2 %O	PUSH,2,I⊕¦DEL	82
83. 3 (	PUSH,2,IND(¦DEL¦INDX,1	84
STATE NUMBER: 84		
84. 1 :R	PUSH,1,I⊕¦DEL	85

```
                                FINITE-STATE-MACHINE RULES
                                MATCH - ACTION - NEXT_STATE

STATE NUMBER:  85
  85. 1  (                      PUSH,2,IND(¦DEL¦ADDON,1@¦POP,1          86
  85. 2  ),,IND(                ADDON,1@¦POP,1¦DEL¦POP,2                86

STATE NUMBER:  86
  86. 1  ),,IND(                DEL¦POP,2                               86
  86. 2                                                                 87

STATE NUMBER:  87
  87. 1  )                      PUSH,2,I@¦DEL                           61
  87. 2  %O                     PUSH,2,I@¦DEL                           82

STATE NUMBER:  88

STATE NUMBER:  90
  90. 1  :R                     PUSH,1,C@;V;¦DEL                        91

STATE NUMBER:  91
  91. 1  (                      PUSH,2,IND(¦DEL¦INDX,1                  92
  91. 2                                                                 94

STATE NUMBER:  92
  92. 1  :R                     PUSH,1,C@;V;¦DEL                        93

STATE NUMBER:  93
  93. 1  (                      PUSH,2,IND(¦DEL¦ADDON,1@¦POP,1          92
  93. 2  ),,IND(                ADDON,1@¦POP,1¦DEL¦POP,2                94

STATE NUMBER:  94
  94. 1  ),,IND(                DEL¦POP,2                               94
  94. 2  )                      PUSH,2,I@¦DEL                           61

STATE NUMBER:  95

STATE NUMBER: 100
 100. 1  ,,{                    POP,2                                   101
 100. 2                         PUSH,2,SUBR:52                          111

STATE NUMBER: 101
 101. 1  ,,{                    POP,2                                   102
 101. 2                         PUSH,2,SUBR:52                          110

STATE NUMBER: 102
 102. 1  ,,{                    POP,2                                   103
 102. 2                         PUSH,2,SUBR:52                          109

STATE NUMBER: 103
 103. 1  ,,{                    POP,2                                   104
 103. 2                         PUSH,2,SUBR:52                          108
```

FINITE-STATE-MACHINE RULES
MATCH - ACTION - NEXT_STATE

STATE NUMBER: 104		
104. 1 ..(	POP,2	105
104. 2	PUSH,2,SUBR:52	107
STATE NUMBER: 105		
105. 1	PUSH,2,SUBR:52	106
STATE NUMBER: 106		
106. 1	PUSH,2,(	107
STATE NUMBER: 107		
107. 1	PUSH,2,(	108
STATE NUMBER: 108		
108. 1	PUSH,2,(	109
STATE NUMBER: 109		
109. 1	PUSH,2,(	110
STATE NUMBER: 110		
110. 1	PUSH,2,(	111
STATE NUMBER: 111		
111. 1		55
STATE NUMBER: 112		
112. 1 ..(	POP,2	113
112. 2	PUSH,2,SUBR:60	123
STATE NUMBER: 113		
113. 1 ..(	POP,2	114
113. 2	PUSH,2,SUBR:60	122
STATE NUMBER: 114		
114. 1 ..(	POP,2	115
114. 2	PUSH,2,SUBR:60	121

FINITE-STATE-MACHINE RULES
MATCH - ACTION - NEXT_STATE

STATE NUMBER: 115		
115. 1 ..(	POP,2	116
115. 2	PUSH,2,SUBR:60	120
STATE NUMBER: 116		
116. 1 ..(	POP,2	117
116. 2	PUSH,2,SUBR:60	119
STATE NUMBER: 117		
117. 1	PUSH,2,SUBR:60	118
STATE NUMBER: 118		
118. 1	PUSH,2,(	119
STATE NUMBER: 119		
119. 1	PUSH,2,(	120
STATE NUMBER: 120		
120. 1	PUSH,2,(	121
STATE NUMBER: 121		
121. 1	PUSH,2,(	122
STATE NUMBER: 122		
122. 1	PUSH,2,(	123
STATE NUMBER: 123		
123. 1		55
STATE NUMBER: 124		
124. 1 ..(	POP,2	125
124. 2	PUSH,2,SUBR:66	135
STATE NUMBER: 125		
125. 1 ..(	POP,2	126
125. 2	PUSH,2,SUBR:66	134
STATE NUMBER: 126		
126. 1 ..(	POP,2	127
126. 2	PUSH,2,SUBR:66	133
STATE NUMBER: 127		
127. 1 ..(	POP,2	128
127. 2	PUSH,2,SUBR:66	132
STATE NUMBER: 128		
128. 1 ..(	POP,2	129
128. 2	PUSH,2,SUBR:66	131

FINITE-STATE-MACHINE RULES
MATCH - ACTION - NEXT_STATE

```
STATE NUMBER: 129
 129. 1                          PUSH,2,SUBR:66                  130

STATE NUMBER: 130
 130. 1                          PUSH,2,(                        131

STATE NUMBER: 131
 131. 1                          PUSH,2,(                        132

STATE NUMBER: 132
 132. 1                          PUSH,2,(                        133

STATE NUMBER: 133
 133. 1                          PUSH,2,(                        134

STATE NUMBER: 134
 134. 1                          PUSH,2,(                        135

STATE NUMBER: 135
 135. 1                                                           55

STATE NUMBER: 136

STATE NUMBER: 140
 140. 1  (                       DEL                             141

STATE NUMBER: 141
 141. 1  (                       DEL                             142

STATE NUMBER: 142
 142. 1  @VIRT                   PUSH,1,I@¦DEL                   143

STATE NUMBER: 143
 143. 1  )                       DEL¦BYENT¦POP,1                 144

STATE NUMBER: 144
 144. 1  )                       DEL                              53
 144. 2                          PUSH,2,SUBR:145                  11

STATE NUMBER: 145
 145. 1                          ATTBY¦POP,1                     146

STATE NUMBER: 146
 146. 1  )                       DEL                              53
 146. 2  %O                      DEL¦PUSH,2,SUBR:145              11

STATE NUMBER: 147
MACHINE DEFINITION COMPLETE
```

(A SAMPLE SESSION OF A VERY BASIC EXAMPLE)

IN THE FOLLOWING PAGES, A SAMPLE TERMINAL SESSION IS ILLUSTRATED.
LINES WHICH BEGIN WITH '$$$' ARE USER INPUT LINES, AND
LINES WITH TEXT ENCLOSED IN A SET OF PARENTHESES
ARE NEITHER USER INPUT NOR PROGRAM OUTPUT, THEY ARE
DESCRIPTIONS OF THE ILLUSTRATION.
ALL OTHER LINES ARE PROGRAM OUTPUT; HOWEVER, THE PRINTING OF
THE TOKEN CHAIN, THE MACHINE ACTIVITIES, THE EXECUTION TREE, AND
THE ENTITY SET TABLE MAY BE SUPPRESSED IN ACTUAL USE.

STATUS: CMS FILE "FILE ZZZ" CONTAINS THE FOLLOWING FOUR
LINES:

```
   DEFINE HEAVY AS WEIGHT > 300 ;
   ADHOC IQ AS 1/GPA * 2500 ;
   RETRIEVE ( ( EMPLOYEE ) WHERE (HEAVY AND INDEX > 100 OR  IQ > 150) )
              BY (( VO ) NAME) ;
```

```
(IN CMS)
R;
$$$                                   LOAD USINT (START NODUP
(SCREEN REFRESHED)
```

```
              ***INFOPLEX DATA BASE MACHINE***
TYPE "VIR" FOR VIRTUAL INFORMATION PROCESSOR
TYPE "REAL" FOR REAL INFORMATION PROCESSOR
:
$$$                                    VIR
(SCREEN REFRESHED)
```

```
                        ***TRANSACTION BUFFER***
 0                       ***NO MORE***
 **
 **
 **
 **

                        ***EXECUTION BUFFER***
                         ***NO MORE***
 **
 **
 **
 **
 **
 **
 **
 **
 **
 **
 **
$$$                                  FINPUT ZZZ
(SCREEN REFRESHED)
```

READING TRANSACTION BUFFER FROM CMS FILE:
"FILE ZZZ"
OLD TRANSACTION BUFFER CONTENT DELETED
(SCREEN REFRESHED)

```
                          ***TRANSACTION BUFFER***
 0 DEFINE HEAVY AS WEIGHT > 300 ;
 1 ADHOC IQ AS 1/GPA * 2500 ;
 2 RETRIEVE ( { EMPLOYEE } WHERE (HEAVY AND INDEX > 100 OR  IQ > 150) )
 3             BY ({ VO } NAME) ;
 4                          ***NO MORE***
**
                           ***EXECUTION BUFFER***
                            ***NO MORE***
**
**
**
**
**
**
**
**
**
**
**
:
$$$                                    RUNTRANS
(SCREEN REFRESHED)
```

```
@HEAVY@ DEFINITION: SAVED_IN_MAIN
TYPE "ENTER" KEY TO CONTINUE
***                                  "ENTER" KEY
(SCREEN REFRESHED)
```

```
@IQ@ DEFINITION: SAVED_IN_ADHOC
TYPE "ENTER" KEY TO CONTINUE
$$$                                    "ENTER" KEY
(SCREEN REFRESHED)
```

```
RETRIEVE ( { EMPLOYEE } WHERE (HEAVY AND INDEX > 100 OR  IQ > 150) )
           BY ({ VO } NAME) ;
(LIST OF INPUT TOKENS)
TEST_TOKEN =$RETRIEVE$$
TEST_CLASS =$:B$$
TEST_TOKEN =$($$
TEST_CLASS =$:D$$
TEST_TOKEN =${$$
TEST_CLASS =$:D$$
TEST_TOKEN =$EMPLOYEE$$
TEST_CLASS =$:R$$
TEST_TOKEN =$}$$
TEST_CLASS =$:D$$
TEST_TOKEN =$WHERE$$
TEST_CLASS =$:R$$
TEST_TOKEN =$($$
TEST_CLASS =$:D$$
TEST_TOKEN =$WEIGHT$$
TEST_CLASS =$:R$$
TEST_TOKEN =$>$$
TEST_CLASS =$:B$$
TEST_TOKEN =$      300$$
TEST_CLASS =$:A$$
TEST_TOKEN =$AND$$
TEST_CLASS =$:B$$
TEST_TOKEN =$INDEX$$
TEST_CLASS =$:R$$
TEST_TOKEN =$>$$
TEST_CLASS =$:B$$
TEST_TOKEN =$      100$$
TEST_CLASS =$:A$$
TEST_TOKEN =$OR$$
TEST_CLASS =$:B$$
TEST_TOKEN =$        1$$
TEST_CLASS =$:A$$
TEST_TOKEN =$/$$
TEST_CLASS =$:B$$
TEST_TOKEN =$GPA$$
TEST_CLASS =$:R$$
TEST_TOKEN =$+$$
TEST_CLASS =$:B$$
TEST_TOKEN =$     2500$$
TEST_CLASS =$:A$$
TEST_TOKEN =$>$$
TEST_CLASS =$:B$$
TEST_TOKEN =$      150$$
TEST_CLASS =$:A$$
TEST_TOKEN =$)$$
TEST_CLASS =$:D$$
TEST_TOKEN =$)$$
TEST_CLASS =$:D$$
```

```
TEST_TOKEN =$BY$$
TEST_CLASS =$:B$$
TEST_TOKEN =$($$
TEST_CLASS =$:D$$
TEST_TOKEN =${$$
TEST_CLASS =$:D$$
TEST_TOKEN =$VO$$
TEST_CLASS =$:R$$
TEST_TOKEN =$}$$
TEST_CLASS =$:D$$
TEST_TOKEN =$NAME$$
TEST_CLASS =$:R$$
TEST_TOKEN =$)$$
TEST_CLASS =$:D$$
TEST_TOKEN =$;$$
TEST_CLASS =$:D$$
MACHINE DEFINITION COMPLETE
(END OF TOKEN CHAIN)
(FINITE STATE MACHINE DEFINED)
```

```
(TRACE OF FINITE STATE TRANSITIONS)

TRANSIT: STATE =     1
INPUT =$RETRIEVE$$
CLASS =$:B$$
STK#1 =$@BOS$$
STK#2 =$@BOS$$
MATCHING =$RETRIEVE$$
FIND_MATCH =$RETRIEVE$$
FIND_ITEM =$RETRIEVE$$
FIND_CLASS =$:B$$
SEP_ITEM =$RETRIEVE$$
FIND_MATCH =$$$
FIND_ITEM =$@BOS$$
FIND_CLASS =$$$
FIND_MATCH =$$$
FIND_ITEM =$@BOS$$
FIND_CLASS =$$$
ACTING =$PUSH,2,I@$$
ACTING =$DEL$$
ACTING =$PUSH,2,SUBR:2$$

TRANSIT: STATE =    47
INPUT =$($$
CLASS =$:D$$
STK#1 =$@BOS$$
STK#2 =$SUBR:2$$
MATCHING =$($$
FIND_MATCH =$($$
FIND_ITEM =$($$
FIND_CLASS =$:D$$
SEP_ITEM =$($$
FIND_MATCH =$$$
FIND_ITEM =$@BOS$$
FIND_CLASS =$$$
FIND_MATCH =$$$
FIND_ITEM =$SUBR:2$$
FIND_CLASS =$$$
ACTING =$DEL$$

TRANSIT: STATE =    48
INPUT =${$$
CLASS =$:D$$
STK#1 =$@BOS$$
STK#2 =$SUBR:2$$
MATCHING =${$$
FIND_MATCH =$($$
FIND_ITEM =${$$
FIND_CLASS =$:D$$
SEP_ITEM =${$$
FIND_MATCH =$$$
FIND_ITEM =$@BOS$$
FIND_CLASS =$$$
FIND_MATCH =$$$
```

```
FIND_ITEM =$SUBR:2$$
FIND_CLASS =$$$
ACTING =$PUSH,2,I@$$
ACTING =$DEL$$

TRANSIT: STATE =    54
INPUT =$EMPLOYEE$$
CLASS =$:R$$
STK#1 =$@BOS$$
STK#2 =${$$
MATCHING =$:R$$
FIND_MATCH =$:R$$
FIND_ITEM =$EMPLOYEE$$
FIND_CLASS =$:R$$
FIND_MATCH =$$$
FIND_ITEM =$@BOS$$
FIND_CLASS =$$$
FIND_MATCH =$$$
FIND_ITEM =${$$
FIND_CLASS =$$$
ACTING =$PUSH,1,I@$$
ACTING =$DEL$$

TRANSIT: STATE =    49
INPUT =$}$$
CLASS =$:D$$
STK#1 =$EMPLOYEE$$
STK#2 =${$$
MATCHING =$),@VIRT$$
FIND_MATCH =$}$$
FIND_ITEM =$}$$
FIND_CLASS =$:D$$
SEP_ITEM =$}$$
FIND_MATCH =$@VIRT$$
FIND_ITEM =$EMPLOYEE$$
FIND_CLASS =$$$
SEP_ITEM =$V0,V1,V2,V3,V4,V5,V6,V7,V8,V9$$
SEP_ITEM =$V1,V2,V3,V4,V5,V6,V7,V8,V9$$
SEP_ITEM =$V2,V3,V4,V5,V6,V7,V8,V9$$
SEP_ITEM =$V3,V4,V5,V6,V7,V8,V9$$
SEP_ITEM =$V4,V5,V6,V7,V8,V9$$
SEP_ITEM =$V5,V6,V7,V8,V9$$
SEP_ITEM =$V6,V7,V8,V9$$
SEP_ITEM =$V7,V8,V9$$
SEP_ITEM =$V8,V9$$
SEP_ITEM =$V9$$
MATCHING =$}$$
FIND_MATCH =$}$$
FIND_ITEM =$}$$
FIND_CLASS =$:D$$
SEP_ITEM =$}$$
FIND_MATCH =$$$
FIND_ITEM =$EMPLOYEE$$
```

178

```
FIND_CLASS =$$$
FIND_MATCH =$$$
FIND_ITEM =$($$
FIND_CLASS =$$$
ACTING =$GENENT$$
ACTING =$DEL$$

TRANSIT: STATE =    50
(.....................)
(.....................)
(.....................)
(MORE OF THE SAME)
(.....................)
(.....................)
(FROM THIS POINT ON ONLY THE TRANSFER FROM STATE TO STATE WILL BE SHOWN)
```

```
TRANSIT: STATE =    30
TRANSIT: STATE =    31
TRANSIT: STATE =    11
TRANSIT: STATE =    14
TRANSIT: STATE =    12
TRANSIT: STATE =    33
TRANSIT: STATE =    34
TRANSIT: STATE =    11
TRANSIT: STATE =    12
TRANSIT: STATE =    35
TRANSIT: STATE =    35
TRANSIT: STATE =    31
TRANSIT: STATE =    11
TRANSIT: STATE =    14
TRANSIT: STATE =    12
TRANSIT: STATE =    33
TRANSIT: STATE =    34
TRANSIT: STATE =    11
TRANSIT: STATE =    12
(.....................)
(.....................)
(MORE OF THE SAME)
(.....................)
(SKIP TO NEAR THE END)
```

```
TRANSIT: STATE =     53
INPUT =$;$$
CLASS =$:D$$
STK#1 =$:        1$$
STK#2 =$SUBR:2$$
MATCHING =$,,SUBR$$
FIND_MATCH =$$$
FIND_ITEM =$;$$
FIND_CLASS =$:D$$
FIND_MATCH =$$$
FIND_ITEM =$:        1$$
FIND_CLASS =$$$
FIND_MATCH =$SUBR$$
FIND_ITEM =$SUBR:2$$
FIND_CLASS =$$$
SEP_ITEM =$SUBR$$

TRANSIT: STATE =      2
INPUT =$;$$
CLASS =$:D$$
STK#1 =$:        1$$
STK#2 =$RETRIEVE$$
MATCHING =$;,,RETRIEVE$$
FIND_MATCH =$;$$
FIND_ITEM =$;$$
FIND_CLASS =$:D$$
SEP_ITEM =$;$$
FIND_MATCH =$$$
FIND_ITEM =$:        1$$
FIND_CLASS =$$$
FIND_MATCH =$RETRIEVE$$
FIND_ITEM =$RETRIEVE$$
FIND_CLASS =$$$
SEP_ITEM =$RETRIEVE$$
(END OF STATE TRANSITIONS)
(INPUT SUCCESSFULLY PARSED)
(EXECUTION TREE AND TABLE ON PAGES TO FOLLOW)
```

(EXECUTION TREE)

```
XTREE_LOC =      1
LABEL = >
CHILD =      2
LINK  =      4
XTREE_LOC =      2
LABEL =
CHILD =      0
LINK  =      3
XTREE_LOC =      3
LABEL =       300:A
CHILD =      0
LINK  =      0
XTREE_LOC =      4
LABEL =          1:NTH
CHILD =      0
LINK  =      0
XTREE_LOC =      5
LABEL = >
CHILD =      2
LINK  =      8
XTREE_LOC =      6
LABEL =
CHILD =      0
LINK  =      7
XTREE_LOC =      7
LABEL =       100:A
CHILD =      0
LINK  =      0
XTREE_LOC =      8
LABEL =          2:NTH
CHILD =      0
LINK  =      0
XTREE_LOC =      9
LABEL = AND
CHILD =      2
LINK  =      1
XTREE_LOC =     10
LABEL =
CHILD =      0
LINK  =      5
XTREE_LOC =     11
LABEL = /
CHILD =      2
LINK  =     14
XTREE_LOC =     12
LABEL =
CHILD =      0
LINK  =     13
```

```
XTREE_LOC =     13
LABEL =          3:NTH
CHILD =      0
LINK  =      0
XTREE_LOC =     14
LABEL =          1:A
CHILD =      0
LINK  =      0
XTREE_LOC =     15
LABEL = *
CHILD =      2
LINK  =     11
XTREE_LOC =     16
LABEL =
CHILD =      0
LINK  =     17
XTREE_LOC =     17
LABEL =      2500:A
CHILD =      0
LINK  =      0
XTREE_LOC =     18
LABEL = >
CHILD =      2
LINK  =     15
XTREE_LOC =     19
LABEL =
CHILD =      0
LINK  =     20
XTREE_LOC =     20
LABEL =       150:A
CHILD =      0
LINK  =      0
XTREE_LOC =     21
LABEL = OR
CHILD =      2
LINK  =      9
XTREE_LOC =     22
LABEL =
CHILD =      0
LINK  =     18
(END OF EXECUTION TREE)
(ENTITY SET TABLE TO FOLLOW)
```

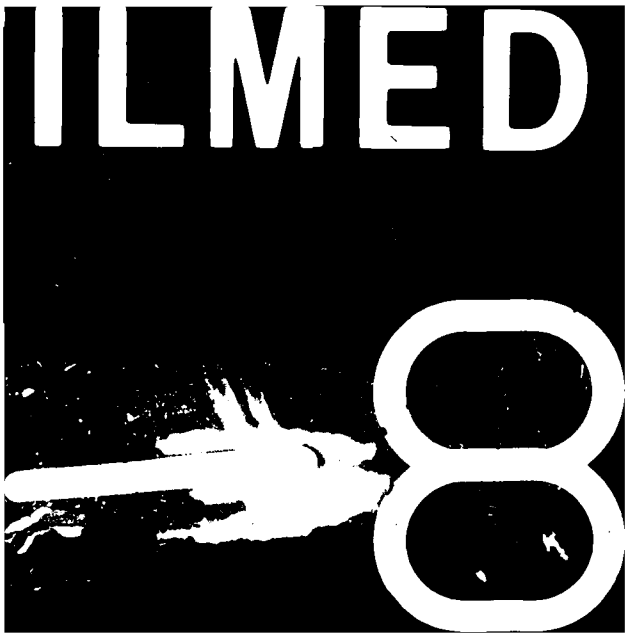
```
ENT_NAME: EMPLOYEE
ENT_DEPTH:      0
ATTRIBUTE:    1
     SING_OCC
     A_PARENT:          0
     USES: WEIGHT
ATTRIBUTE:    2
     SING_OCC
     A_PARENT:          0
     USES: INDEX
ATTRIBUTE:    3
     SING_OCC
     A_PARENT:          0
     USES: GPA
ATTRIBUTE:    4
     SING_OCC
     A_PARENT           0
     USES: NAME
TYPE "ENTER" KEY TO CONTINUE
$$$                                    "ENTER" KEY
(SCREEN REFRESHED)
```

178

```
                          ***TRANSACTION BUFFER***
 0 DEFINE HEAVY AS WEIGHT > 300 ;
 1 ADHOC IQ AS 1/GPA * 2500 ;
 2 DEFINE INDEX AS SALARY + ID * DEPTNUM - WEIGHT / (SECNUM ! 2) ;
 3 RETRIEVE ( ( EMPLOYEE ) WHERE (HEAVY AND INDEX > 100 OR  IQ > 150) )
 4            BY (( VO ) NAME) ;
                          ***EXECUTION BUFFER***
                           ***NO MORE***
**
**
**
**
**
**
**
**
**
**
**
:
$$$                                    TERMINATE
(SCREEN REFRESHED)
```

R;
(BACK TO CMS)

081


ILMED
8